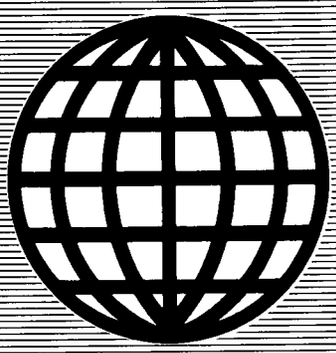


The COMPUTER JOURNAL

Programming - User Support
Applications

Issue Number 48 January / February 1991 $3.95

From The Desk

Fast Math Using Logarithms

Real Computing

Forth and Forth Assembler

Modula-2 and the TCAP

The Z-System Corner

Adding a Bernoulli Drive to a CP/M Computer

Z Best Software

Review of BDS "Z"

PMATE / ZMATE Macros

The Computer Corner

ISSN # 0748-9331

The Computer Journal

Founder
Art Carlson

Editor/Publisher
Chris McEwen

Technical Consultant
William P. Woodall

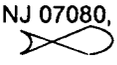
Contributing Editors
Bill Kibler
Tim McDonough
Bridger Mitchell
Clem Pepper
Richard Rodman
Jay Sage

The Computer Journal is published six times a year by *The Computer Journal*, P.O. Box 12, S. Plainfield, NJ 07080. (908) 755-6186

Entire contents copyright © 1991 by *The Computer Journal*. All rights reserved. Reproduction in any form prohibited without express written permission of the publisher.

Subscription rates•$18 one year (6 issues), or $32 two years (12 issues) in the U.S., $24 one year surface in other countries. Inquire for air rates. All funds must be in U.S. dollars on a U.S. bank.

Send subscription, renewals, address changes, or advertising inquires to: *The Computer Journal*, P.O. Box 12, S. Plainfield. NJ 07080, phone (908) 755–6186.



Registered Trademarks

It is easy to get in the habit of using company trademarks as generic terms, but these trademarks are the property of the respective companies. It is important to acknowledge these trademarks as their property to avoid their losing the rights and the term becoming public property. The following frequently used trademarks are acknowledged, and we apologize for any we have overlooked.

Apple II, II+, IIc, IIe, Lisa, Macintosh, DOS 3.3, ProDos; Apple Computer Company. CP/M, DDT, ASM, STAT, PIP; Digital Research. DateStamper, BackGrounder II, Dos Disk; Plu*Perfect Systems. Clipper, Nantucket; Nantucket, Inc. dBase, dBASE II, dBASE III, dBASE III Plus, dBASE IV; Ashton-Tate, Inc. MBASIC, MS-DOS, Windows, Word; MicroSoft. WordStar; MicroPro International. IBM-PC, XT, and AT, PC-DOS; IBM Corporation. Z80, Z280; Zilog Corporation. Turbo Pascal, Turbo C, Paradox; Borland International. HD64180; Hitachi America, Ltd. SB180; Micromint, Inc.

Where these and other terms are used in The Computer Journal, they are acknowledged to be the property of the respective companies even if not specifically acknowledged in each occurrence.

The COMPUTER JOURNAL

Issue Number 48 January / February 1991

Editorial 2

From the Desk 3
A Time for Change
By Art Carlson.

Fast Math Using Logarithms 4
A Technique for Micro-Controllers
By C. V. Palm.

Real Computing 7
Minix 1.5 and X-10 Modules
By Richard Rodman.

Forth and Forth Assembler 9
By Matthew Mercaldo.

Modula-2 and the TCAP 11
Writing Terminal Specific Code
By David L. Clarke.

The Z-System Corner 17
Patching MEX-Plus and The Word, Using ZEX
By Jay Sage.

Adding a Bernoulli Drive to a CP/M Computer 25
By Wayne Sung.

Z Best Software 30
A Look at New Z-System Programs
By Bill Tishey.

Review of BDS "Z" 32
Bringing C and Z-System Together
By Carson Wilson.

PMATE / ZMATE Macros 34
By Clif Kinne.

The Computer Corner 40
By Bill Kibler.

Editorial

By Chris McEwen

Some Things Never Change.

There I was, sitting fat, dumb and happy. The screen was blinking, the drives were working and I hadn't trashed a file for hours. Things were going my way. The telephone rang. It was Art Carlson.

This was a surprise. Mind you, I've spoken to Art on many occasions, but only when he wasn't expecting my call. I find Art to be a true resource of good advice. He may not tell me what I want, but has always says what I need to hear. You can see my surprise: students call teachers, not the other way around.

Art related a conversation he and Jay Sage had the previous night. He was looking for someone to take the reins of *The Computer Journal.* Jay mentioned my name. Would I be willing to be *TCJ*'s new publisher? By this time, I was no longer sitting fat, dump and happy. I had gracefully fallen off my chair!

What do you say to someone you truly admire, someone you look to and think, "if I live to be a hundred, I'll have learned half that this man has already forgotten," when he asks if you will step in and take his place? "No! No, I can't!" This was just too much. Who can walk in Art's shoes without looking like a five year old mimicking his father? But Art was serious; his health required that he take a break.

My mind was racing. Could I do it? No, truthfully, I cannot. It was time for a telephone call of my own. Jay Sage is another person who I look to for good advise. I told him of Art's call and my concerns.

If you've ever had a conversation with Jay, you know he is a good listener. He heard me out and then calmly said, "You won't do it alone. There is more to *TCJ* than one person. It lives through the involvement of many: the authors, the readers and the advertisers. Your question should be, 'Can *we* do it?' And I say, 'We can!'"

This started two weeks of turmoil. Could we do it? I determined that if we could not assure two things that I would not accept Art's offer:

1. We must maintain the content quality. Art and Jay convinced me we would and offered editorial assistance. The articles are the product of the readers. This is *TCJ*'s traditional strength.

2. We must maintain the quality of the journal itself. One of our readers, Bill Woodall, lives nearby. Bill is a publisher as well as software developer and jumped right in to help. In fact, Bill has already earned a place on the masthead. Frankly, this issue would have never seen light of day without his help. Thank you, Bill!

I have been reminded of the difference between a magazine and a journal. A magazine is put out by the large staff of a large corporation, serving a large number of people who are called customers. A journal is written by its readership, assembled and produced by a few and sent out to people who are called friends.

So there you have it, friends. Art will be staying with us as adviser and author. I look forward to his commentary on the future of computing. But in the end, *TCJ* is not Art and Art is not *TCJ.* We—all of us—are *TCJ* and always have been.

Some things will never change.

Is Our Slip Showing?

You know, I always had great respect for Art. His insight is on target so often that one wonders where he keeps the crystal ball. But I never guessed the sheer *work* involved in getting this rag put to bed! Critical readers will surely note errors in layout and typesetting. It seems every time I look at a page that I have been over a dozen times, another problem jumps out at me. Your patience is appreciated. As they say, practice makes perfect. In my case, maybe not perfect, but surely things will settle down.

In the meantime, a few things have changed due to the different equipment Art and I have. The body type is changed from Bitstream Dutch to Adobe Palatino. Type preference is a very personal thing. I find Palatino a little more open. We may see some other changes. It would be nice to have some photographs of the projects described. And maybe a shot of the authors. Maybe. Maybe not. Carson Wilson looks decent, Jay Sage is okay. But it would be no service to run my photo! Anyway, you be the judge. Personal, outrageous opinions are solicited.

The Family That Writes Together...

Art talks about "a time for change" in his "From The Desk" column. Something I hope never changes is the quality of articles our readers submit. Look through your back issues. None of the articles you see were written by professional writers. So where does the quality come from? Commitment! Whether it be to an operating system, language or hardware platform, we are a particularly enthusiastic group. It shows in what we write. I sincerely seek your guidance in the direction of future issues. And the best way for you to influence that direction is in your submissions.

How does one submit an article for publication? We are educated, literate people here, so you need not write to the "lowest common denominator" as so many editors would have you do. At the same time, be sufficiently clear in your explanations. If you remember that your audience comes from several computer disciplines and some may need terms defined, then you should do fine. Seek your own writing style. Some are folksy. Some are straight and to the point. But

From The Desk...

And Now, a Word from Our Founder

By Art Carlson

A Time for Change

I started *TCJ* almost eight years ago because other publications did not provide the information I needed. Many things have changed during the ensuing period, but one thing has remained constant—other publications still don't provide the necessary information. I have enjoyed publishing *TCJ*, and have especially appreciated the many fine people that I have contacted through *TCJ*. But, the past few years have been a struggle due to my personal health problems, and *TCJ* deserves more effort and energy than I can provide—it is time to get help.

I spent a lot of time thinking about the type of person who could continue *TCJ* from its present base, and expand it without ruining what we all have worked so hard to achieve. It had to be someone who was interested in our areas of computing, but who was also aware of the business aspects of marketing and production. After talking to Jay Sage and others, I contacted Chris McEwen. Chris is a Z-NODE sysop, has a good understanding of printing and promotion, and had just purchased a '386 system and a laser printer with the intention of getting into the desk top publishing business. Little did he know just how fast he would be in business!

Chris isn't going to have to do it by himself. No one could. We're all going to have to help, because *TCJ*'s one great strength is the wide and varied viewpoint from many different people. I need to take a short break to catch my breath, but intend to continue writing for *TCJ*. I'll also be available to answer questions, but Chris will make the decisions.

I have several lists of long delayed hardware and software projects and I am looking forward to being able to spend more time at the bench and on programming. I'll write about what I am doing, and Chris might publish some of it if he feels that it will interest the readers. Along this line, Chris needs to hear from *you* in order to know what you want. I believe that he will be including a survey in one of the next issues, but don't wait for that if you have something to say.

---

everyone writes from a knowledge base that cannot be duplicated.

Submit articles on either disk or as an upload to Socrates Z-Node. Aim for an article length between 10k and 25k in length in ASCII format. We can also handle data files from most of the major word processing systems. But, please, do not try to format the file except when necessary. Writing style is your job; page layout and publication style is mine.

Continued Page 24

His BBS will make it very easy for you to contact him—use it!

Most of my work will be on small hardware and software projects involving microcontrollers, logic chips, linear devices (transistors, A/D, etc.), LCD displays, motors, monitor and debugging software, etc. Let Chris know if any of this interests you.

More on Hard Drive Problems

I mentioned that I had added a fan to a Seagate 225 drive in an attempt to cure heat related problems. One of our readers used another approach to solve his problems with a Seagate 251.

The drive would work fine when it was first turned on, but disk read errors would creep in after about a half hour. Reformatting did not help. It obviously needed some heatsinking.

On inspection he noticed one chip with lots of pins and a heatsink. The engineers at Seagate cleverly use the ground plane as a thermal path, but both it and the device face in towards the drive! Not much help.

He carefully unsoldered the chip and bent the pins backwards, then mounted the device to the outside of the circuit board, with a bona-fide aluminum heat sink. Of course this causes the whole thing to protrude a bit, but there is plenty of room in the cabinet. Since this change, he has had no problems with the drive.

This is a good example of problem solving and hardware hacking.

Harris RTX Dies

Harris has just completed an expensive promotional campaign and contest for their RTX Forth chip. Now, they have celebrated the contest by dropping the product line. It would be very interesting to hear the inside information on their reasons.

Embedded Controllers

We have a number of interesting articles in progress. Tim McDonough is hard at work on an 8051 project incorporating A/D conversion using the ADC0808, which he expects to have ready for the next issue. Matthew Mercaldo (one of the Harris RTX contest winners) is working on his stepper motor series, and will be using the New Micros F68HC11 Forth chip. It is likely that we will have an article on using power MOSFETs as high-current drivers, and I will have something as soon as I get things sorted out—after all, one of the primary reasons for this change is so that I can spend more time on the bench and writing.

Contact Chris if you have any ideas about articles.•

Fast Math Using Logarithms

A Technique for Micro–Controllers

By C. V. Palm

Years ago, before the pocket calculator, fast multiplication and division was performed by hand, using log tables. Slide rules also use log scales to perform these and other functions very quickly, albeit with some limitation to accuracy. If a simple slide rule can do it, why not a micro controller?

The basic principle is that when the log of a value is added to the log of another value, the antilog of the sum reflects the product of the two values. If subtracted instead of added, we have the quotient of the values. In other words the antilog of LOG(a)+LOG(b) equals a*b, antilog of LOG(a)-LOG(b) equals a/b, and -LOG(a) reflects 1/a. Exponents and roots can also be found a LOG(a)*n becomes a^n, and LOG(a)/n becomes $\sqrt[n]{a}$.

So, to perform a division for example we simply have to do three conversions and one subtraction. The speed of this division will then mainly depend on the time taken to do a conversion.

But first, what is a log? Simply put it is the power to which the number base has to be raised to obtain the value. Thus in base 10 (decimal), $LOG_{10}100=2$ as $10^2 = 100$. Logs, as other math functions, can be in any number base. Computer languages such as BASIC use natural logs or LOG_e meaning LOG to base e. Published log tables normally use base 10 or LOG_{10}. To calculate logs in a micro controller we shall use base 2 (binary)or LOG_2 as the controller is much more at home there.

To convert a log from any base to any other base i.e. LOG base b of the value X to LOG base a of the same value, use the formula $LOG_a(X)=LOG_b(X) / LOG_b(a)$. For example, your computer will give you the natural log or LOG_e for the value x from the expression LOG(X). To find the $LOG_2(X)$, use the expression LOG(X)/LOG(2).

Armed with this information we can find the binary logs for any value. The problem is that the type of processors we will use do not have and built in log functions, and in most cases there is not enough space to provide full log tables. To actually calculate the logs with the available instructions, we have to examine the binary logs more closely. Some simple examples are: $LOG_2(4)=2$, $LOG_2(8)=3$ and $LOG_2(16)=4$. As expected these are powers of two since $2^2=4$, $2^3=8$, $2^4=16$.

A log consists of two parts: an integer characteristic and a fractional mantissa. The precision of a result depends on how many digits there are in the mantissa. In the above examples the characteristics are 4, 8 and 16 respectively. When the log is taken from an integer power of the base, the mantissa is always zero, so the results expressed to 4 digits precision are 4.0000, 8.0000 and 16.0000. LOG(1) equals 0.0000 and logs between 0 and 1 are negative i.e. the binary log of 0.1_b (0.5_d) equals -1.0000. LOG(0) is illegal, like trying to divide by 0.

Matters become more complex when we want a log that is not a power of 2. Let's try $LOG_2 7$ which is 2.8074 in decimal. The fractions to four digit precision equals 0.8074. Now, decimal fractions are frowned upon by our processor, so 8074/10000 equal 52914/65536 or CEB2/10000 in hex. Thus $LOG_2 7$ expressed in hex becomes $2.CEB2_h$, 5h3 characteristic is 02_h and the mantissa $0.CEBA_h$. I will use hex representation rather than binary (easier to type). The result was obtained from the BASIC's LOG function in my computer, but we shall now try to calculate binary logs without any such aid.

The characteristic is easy. The following table shows binary representation for logs 4, 7, 8, 16.

Number	4	7	8	16
Binary value	00000100	00000111	00001000	00010000
Characteristic	2	2	3	4

As you can see, the characteristic is equal to the position of the most significant bit that is set in the argument. This means that an 8-bit characteristic can actually represent a 255-bit value which is rather astronomical! To calculate the characteristic in software, first initialize it to the number of bits used in the argument, then decrement it while shifting the argument left until a 1 is shifted out. Let's see what happens when we try for $LOG_2 7$. As we are using an 8-bit argument we initialize the characteristic to 8.

```
char=char-1 (7) shift 00000111 = 00001110 cy=0
char=char-1 (6) shift 00001110 = 00011100 cy=0
char=char-1 (5) shift 00011100 = 00111000 cy=0
char=char-1 (4) shift 00111000 = 01110000 cy=0
char=char-1 (3) shift 01110000 = 11100000 cy=0
char=char-1 (2) shift 11100000 = 11000000 cy=1
```

When carry is set, char contains the characteristic 02. In 8048 code this can be performed as follows. CHAR and ARGUMENT are registers.

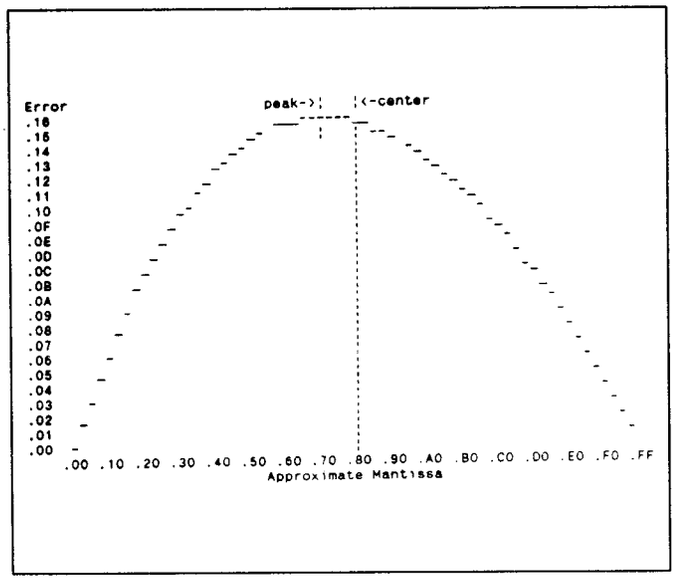
```
BITS    EQU  8              ;for an 8-bit argument
        MOV  CHAR,#BITS
        MOV  A,ARGUMENT
        JZ   ERROR          ;Can't do LOG(0)
        CLR  C              ;clear cy to shift in a 0
LOOP
        DEC  CHAR
        RLC  A              ;CY <- Accumulator <- 0
        JNC  LOOP           ;On exit CHAR=characteristic
```

Don't discard the shifted ARGUMENT (now 11000000_b or $0.C0_h$, we'll use that to calculate the mantissa. It is already a rough approximate if we put a hexadecimal point in front as $0.C0_h$. What we have is a linear approximation of the logarithmic curve. The real mantissa is $0.CEB2_h$ so it's accurate to 4 bits. This might be enough for some applications, but we can improve on it.

The error in this case was $0.0EB2_h$. It will be zero when the mantissa, as calculated above, becomes zero or infinitely large. A

mantissa calculated to $0.FF_h$ is actually $0.FF47_h$, an error of only 0.0047_h. Try to calculate $LOG_2 1FF_h$ in 16 bits. It will result in $8.FF00_h$, but the real value in $8.FF47_h$. The worst case error will occur with a mantissa of around 0.71_h with an error of over 0.16_h.

The good news is that this error is consistent, regardless of what the characteristic comes to. Thus any calculated mantissa will have a specific error that can be added to the result, improving the accuracy. This error can be depicted as a curve, as shown in Figure 1.



The way out of this problem is to use a correction table. The number of entries, and the number of bits in each entry will decide the accuracy of the result. A 16 entry 8-bit table with each entry corresponding to the 4 MS bits of the calculated mantissa will decrease the error several times, costing only 16 bytes of space. With a 128 entry 8-bit table, the error is reduce to a small fraction of a percent.

Let's try the value $B9_h$ as this will produce a worst case error in the mantissa. Binary reproduction of $B9_h$ is 10111001_b. Performing the shifts in an 8-bit environment, the first shift will produce a carry, leaving CHAR as 7 and ARGUMENT as 01110010_b or 72_h i.e. a calculated mantissa of 0.72_h making $LOG_2 B9_h$ 7.72_h. $LOG_2 B9_h$ is actually 7.8808_h, an error of 0.1608_h. A 16 entry error table could read:

Entry #	mant MSB	err +	Entry #	mant MSB	err +	Entry #	mant MSB	err +	Entry #	mant MSB	err +
0	.08	.03	1	.18	.09	2	.28	.0E	3	.38	.11
4	.48	.14	5	.58	.15	6	.68	.16	7	.78	.16
8	.88	.15	9	.98	.14	A	.A8	.12	B	.B8	.10
C	.C8	.0D	D	.D8	.0A	E	.ED	.06	F	.F8	.02

We arrived at a mantissa of 0.72_h. By ignoring the point and the four LS bits we get 7. This will be our entry #. Add in error value from entry #7 (0.16_h) to obtain a corrected mantissa of 0.88_h. $LOG_2 B9_h$ is now 7.88_h, not too bad. For $LOG_2 7$ we had $2.C0_h$. Add in error entry #C of $0.0D_h$ and we get $2.CD_h$ for an actual value of $2.CEB2_h$. As $0.C0_h$ is halfway between two entries the error is worse. Maximum error is 0.03_h for mantissa of $0.0F_h$.

The routine in 8048 code:

```
        ORG   300h              ;put table at start of page
TABLE:
        DB    03h,09h,0Eh,11h,14h,15h,16h,16h
        DB    15h,14h,12h,10h,0Dh,0Ah,06h,02h
CORRECT:
        MOV   A,MANTISSA
        SWAP  A                 ;Swap upper and lower nibble
        ANL   A,#0Fh            ;A=entry number
        MOVP  A,@A              ;Get contents of entry into A
        ADD   A,MANTISSA        ;A=corrected mantissa
;Test for mantissa overflow, or set the last entry to 00
;to prevent this.
        JNC   NO_OVERFLOW
        MOV   A,#0FFh           ;Mantissa should be close to 0ffh
```

There are several methods that can be used to obtain better accuracy. We could simulate a 32 entry table by averaging two adjoining entries where the mantissa falls in between. For a mantissa of $C0_h$, entries #B for $B8_h$ (0.10_h) and #C for $C8_h$ ($0.0D_h$) could be added and then shifted right once to obtain an error of $0E_h$. $LOG_2 7$ has now come to $2.CE_h$. We are getting closer. The table entry points may be modified to facilitate this. With #C set to error for $0.C0_h$ and #D for $0.D0_h$ instead, we only have to examine bit 3 in the calculated mantissa, and perform the averaging with the next higher entry when set.

My preference where space permits is to use a 128 entry table. The mantissa is shifted right once to obtain the entry. If a carry occurs I can also add in the succeeding entry to the error and right shift the result to perform averaging.

To make up the tables I use ordinary BASIC, first to calculate accurate logs, and then to perform the binary calculations and subtract the results to form an error table. This is then written out as a text file that can be loaded directly into an assembler. In BASIC: ER=65536*(LOG(MANT+256/LOG(2)-8-MANT/256). ER return a 16-bit error fraction for mantissa MANT and should be converted to hex. If $MANT=50_d$ (32_h), ER will be 4068 or 0FE4 when converted to 4-digit hex. The error for 0.32_h is 0.0FE4. This could then become 0.10_h after 7/8 rounding to 2 digits.

Manipulating the LOGS

Now that we know how to calculate binary logs, we can perform simple operations on these logs to achieve the desired functions. Do the mantissas first, then the characteristics including any carry/borrow from the mantissas. Where only 8-bit mantissas are used and the processor supports 16-bit operations, this can be achieved by a single instruction.

The only rule is that the mantissas of both arguments have the same number of bits. If required one mantissa may be padded with zeroes. Perform the operation in signed integer. Subtracting a smaller log from a larger will produce a negative result. This is acceptable. Just keep the characteristic to 8 bits and ignore any carry i.e. 5.65_h - $8.BC_h$ = $FC.A9_h$. This would be referred to as LOG_2-3.57_h but here we'll use two's complement because it's easier to work with.

We found $LOG_2 B9_h$ to be 7.88_h, and $LOG_2 7$ $2.CE_h$. To divide $B9_h$ by 7 we would subtract the two logs as 7.88_h-$2.CE_h$ with a result of $4.BA_h$.

In Z80 code for division:

```
D=characteristic, E=mantissa for divisor
H=characteristic, L=mantissa for dividend

OR   A          ;clear carry
SBC  HL,DE      ;subtract divisor from dividend

H=characteristic, L=mantissa for quotient
```

The 8048 code is a bit more complex as it does not support 16-bit ops nor subtraction.

```
DIVIDE:         ;ARG1=divisor, ARG2=dividend
        MOV  A,MANTISSA_ARG1
        CPL  A
        ADD  A,#1
        MOV  MANTISSA_ARG1,A
        MOV  A,CHARACTERISTIC_ARG1
        CPL  A
        ADDC A,#0
MULTIPLY:       ;ARG1=MLTIPLICAND, ARG2=multiplier
        MOV  A,MANTISSA_ARG2            ;add arguments
        ADD  A,MANTISSA_ARG1
        MOV  MANTISSA_RESULT,A          ;store result
        MOV  A,CHARACTERISTIC_ARG2
        ADDC  A,CHARACTERISTIC_ARG1
        MOV  CHARACTERISTIC_RESULT,A
```

This way the same routine can be used for both functions.

Antilogs

All that remains now is to obtain an antilog of the resulting log (or the exponent). First, we shall reintroduce the error, then reverse the shift routine. In other words, the entire procedure to find a log will now be reversed.

If space permits, we should have a separate error table for the antilog function. Then simply read off the error value from the table entry corresponding to the calculated mantissa and subtract it. However, we can make use of our original table by on of the following methods.

First obtain the error for the mantissa and, without destroying the mantissa, then subtract the error from it. Next obtain the error for the difference and subtract that from the original mantissa. This may be repeated, but normally two passes will give sufficient accuracy.

Again in 8048 code:

```
        MOV  A,MANTISSA
        SWAP A                  ;swap upper and lower nibbles
        ANL  A,#0Fh             ;A=entry number
        MOVP A,@1               ;get contents of entry into A
        CPL
        INC  A                  ;negate error to subtract it
        ADD  A,MANTISSA         ;  from mantissa
```

If a separate antilog error table was used, 'A' would now contain the mantissa with the error reintroduced (after an underflow test if necessary). Otherwise A is a temporary value and a second pass is needed. Continue as shown:

```
        SWAP A                  ;use the temporary mantissa
        ANL  A,#0FH
        MOVP A,@A               ;get the final error into A
        CPL  A
        INC  A                  ;negate it
        ADD  A,MANTISSA         ;and add to original mantissa
                                ;A=mantissa - error
```

Using our previous result of $4.BA_h$: Entry #B is 0.10_h. Subtract that from $0.BA_h$ and we get $0.AA_h$. Now use entry #A (0.12_h and subtract that from $0.BA_h$ instead. The difference is $0.A8_h$ which is then right shifted to produce the final result. A third pass would make no difference as we would use entry #A again.

There is a sneaking way to save a few MC's where a single error table is used. When we reintroduce the error, the mantissa will be somewhat smaller than the result of just subtracting the error value hence the second pass. The error curve is not symmetric and if we negate the mantissa, obtain and add the error and finally negate the result, it also ought to be somewhat smaller. In fact, we have a close representation of the desired value. Try the following:

```
        MOV  A,MANTISSA
        CPL  A
        INC  A          ;negate mantissa
        MOV  MANTISSA,A ; and save
        SWAP A          ;swap upper and lower nibbles
        ANL  A,#0Fh     ;A=entry # for negated mantissa
        MOVP A,@A       ;move error into A
        ADD  A,MANTISSA ;add to negated mantissa
        CPL
        INC  A          ;A=mantissa - error
```

Again using $4.BA_h$, negate BA_h to 46_h. Error for 0.46_h as per entry #4 is 0.14_h. Add this to 0.46_h and we get 0.54_h. Finally negate $0.5A_h$ to $0.A6_h$. Compare that with our previous result of $0.A8_h$ and the actual value which is $0.A79_h$.

Once the error has been reintroduced, we perform a reverse bit shift that will give the final result. When we originally obtained the characteristic, we shifted out one bit that went to the carry. This bit must now be reinserted. Shift one to the right with with the carry set, followed by a further number of shifts with carry reset. Increment the characteristic at each shift until equal to the number of bits in the argument. Function complete.

Shift is shown here for $4.A8_h$, the result of our subtraction.

```
shift 1 -> 10101000=11010100 -> 0     char=char+1 (5)
shift 0 -> 11010100=01101010 -> 00    char=char+1 (6)
shift 0 -> 01101010=00110101 -> 000   char=char+1 (7)
shift 0 -> 00110101=00011010 -> 1000  char=char+1 (8)
```

When char=8 (the number of bits we used), the final shift came to $1A_h$. An interesting point is that if we save the bits shifted out we get a binary fraction of the result i.e. 0.1000_b or 0.8_h, thus the actual result obtained in $1A.8_h$. Compare that with our initial values of $B9_h$ and 07_h. When divided, the quotient equals $1A.6E_h$ to two hexadecimal points. Carry will also reflect the last bit shifted out, thus if the bits are discarded there is still an indication that can be used to perform 7/8 rounding of the result.

8048 code for final shift, not saving any fraction shifted out.

```
    MOV  A,CHAR
    CPL  A
    INC  A              ;negate characteristic
    ADD  A,#BITS        ; plus number of bits
    MOV  COUNTER,A      ; as number of shifts
    MOV  A,MANTISSA
    CLR  C
    CPL  C
LOOP
    RRC  A
    CLR  C              ;clear carry for next shift
    DJNZ COUNTER,LOOP   ;A equals 8-bit final result
```

For negative logs the result will be between 0 and 1 i.e. a binary fraction. We could proceed exactly as for a positive value, but as we will do more shifts than there are bits, the right shifted mantissa will always return 00. We can therefore skip the last eight shifts by halting when the incremented characteristic overflows from FF_h to 00.

Take the inverse for the previous example. Negate $04.BA_h$ to $FB.46_h$. Reintroduce the error for 0.46_h to 0.35_h, then perform the shift on $FB.35_h$.

```
shift 1 -> 00110101=10011010 -> 1     char=char+1 (FC)
shift 0 -> 10011010=01001101 -> 01    char=char+1 (FD)
shift 0 -> 01001101=00100110 -> 101   char=char+1 (FE)
shift 0 -> 00100110=00010011 -> 0101  char=char+1 (FF)
shift 0 -> 00010011=00001001 -> 10101 char=char+1 (00)
```

Continued page 37

REAL COMPUTING

Minix 1.5 and X-10 Modules

By Richard Rodman

Minix 1.5

Minix 1.5 is now available for PCs, Atari ST, Amiga, and Macintosh. It comes with full source code. This is a major update over previous versions of Minix. It comes with four new editors: emacs and vi subset clones, ex and ed. It even includes a spelling checker. It comes with a new manual.

Minix now uses protected mode on the 286 and 386 and makes use of "extended" memory. Terminals can be added on serial ports for additional users; kermit and zmodem are included. The Macintosh version is said to run "under Multifinder"; the Atari version requires a 720K drive; the Amiga version doesn't support hard disks, since there is apparently no standard Amiga hard disk.

A great many new utilities are included, and the C library has been greatly expanded, now including termcap support and curses. Since the system calls are compatible with Unix version 7, about any Unix program could be made to run (as long as it's not too large).

The package can be ordered from Prentice-Hall, or you may find it in your local bookstore. Bookstores may not have all five versions, however. It costs more than it did before, but a lot more comes with it. All versions listed below cost $169. Here are the item numbers to order by phone:

MINIX Version 1.5 for the:	Order Number:
IBM: (5 1/4")	(0-13-585076-2)
IBM: (3 1/2")	(0-13-585068-1)
Amiga	(0-13-585043-6)
Atari	(0-13-585035-5)
Macintosh	(0-13-585050-9)

If you have an older version of Minix, you can send the boot disk from it to them and they'll give you a $60 discount.

I hope to have this up and running shortly and can give you my impressions of it. I have run Minix 1.2, and been pretty impressed with it. Very soon, I also expect to be running Minix on my PC-532.

For those of you designing your own computers or building up Designer's Kit boards or Multibus or VME systems, Minix may be a quick way to get a pretty nice operating system running. All you need is a C cross-compiler, assembler and linker for the processor of your choice on another machine, and you should be able to bring it up.

Bare Metal status report

Work on Bare Metal is still proceeding, albeit slowly. A new command interpreter is almost finished, which is pretty much compatible with MS-DOS'. There have been some big improvements in the C runtime library. Also, I figured out how to use relocatable executables, so forget all of the TPA stuff. Most of the tools have been converted to OS/2 "family applications", which means they'll run either under OS/2 or DOS.

X-10 modules

X-10 has been around a while. You know, the little modules that you plug things into and then you can turn them on and off remotely, by means of 40kHz signals carried through the power lines of your house. Mention X-10 to an electronics fanatic, and you get the sort of reaction that "there must be something wrong with it." There were stories of modules burning out and so on.

Well, X-10 is still around and improving. There are now a variety of modules which allow lamps or appliances to be turned on when motion is detected or when doors or windows are opened. These devices usually send X-10 codes when they detect changes. There is also a chime which rings when send a code.

Of course, the usual printed matter refers to these devices in connection with the terms "burglar alarm" and "security", and these terms are red flags to us computer-literate folks: they cause an immediate "stone-age technology" response. But certainly we can find more creative uses for this stuff, since it's so cheap. For example, for $70 from Heathkit you get an outdoor floodlight with motion detector, a lamp module and the sensor chime.

But the most essential thing is called the X-10 Powerhouse computer interface. It was widely advertised a few years ago, and sells for about $60. This is a little box with a serial port and eight buttons. It can be used as a regular controller. It's connected to your computer by means of a 600 baud serial link. Your computer can turn devices on or off or dim lights, and set up to 256 timers for timed commands.

But there are two neat aspects to this box. First, you can program it, then unplug it and take it somewhere else! It has a little battery in it that backs it up during power failure. Once programming is complete, it can be disconnected; there's no absolute need to tie up your serial port except while programming.

The other, neater, aspect to it is that any X-10 commands which come across the line are converted to status messages that the computer can see. Thus, the computer can "listen in" to commands issued by manual controllers, wireless controllers, or even the motion detector or the door/window sensor. This gives the computer remote "eyes" as well as "hands."

Just think about the possibilities. How about a "door answering machine"? When someone walks up to your door when you're not home, a recording begins: "We're sorry, but

we can't come to the door right now. After the beep, please put whatever you have in the bin which will open..." How about having your computer turn on your (quiet) printer in the middle of the night to print listings, and turn it back off when they're finished? How about having the computer turn off the power to the monitor when no one is in the room with it? How about sending in *your* great ideas to *TCJ*, attn. Rube Goldberg Dept. [Ed: Rube has asked me to forward his mail to Richard. We will have a mail truck standing by].

There is an X-10 interest group on Genie. But the box has a relatively simple programming interface, which is well-documented. I wrote a simple C program for controlling the X-10 Powerhouse, which is on the BBS or available from *TCJ*. The program has the following commands; of course you can use subroutines from it in your great programs:

```
Turning a lamp on, off, or dim:
   turn <lamp-number>  on
                       off
                       dim <0 to 15>

Setting a timer:
   timer <event-number> <day> <hr>:<min> <lamp-number> on
                                                       off
                                                       dim
 is one of: today, tomorrow, everyday, sun, mon, tues, etc.

Clearing a timer:
   timer <event-number> clear

Miscellaneous:
   set   - Sets time of box to computer's time
   show  - Shows time of box
   quit  - exit program
```

For very large numbers of devices, X-10 is impractical, so for large-scale industrial automation, it probably isn't the best solution. Also, X-10 is too slow for real-time process control. But there is a broad spectrum of automation applications that X-10 should be excellent for.

A proposal

The X-10 device above is one example of an application for "background processing" on a computer. I can think of lots of others. What I'd really like is a board which could accumulate and/or process data even while the computer was turned off, allowing me to access the data later.

Let's design the board. First, it should have batteries which trickle-charge when the computer is turned on, but operate the board when the computer is turned off. A clever interface allows the computer to communicate with the board. The board has a CMOS processor, such as a 80C31, with a 27C256 PROM and a 32-kbyte CMOS RAM. There is a clever low-power RS-232 driver and/or parallel port; ideally, interfaces would be added on on piggy-boards which would go between the processor and the connector.

One application would be recording a series of temperature readings. Temperature sensors would be located remotely with clocked serial A/D converters. Another application would be a mini-bulletin board to collect mail messages from a modem. Yet another, to monitor the AC line voltage. There are thousands of possibilities, all really opened up because the operation of the board is completely independent of the computer, even of its power cyclings, yet with the economy and fast communication of an add-in board product.

How will you program it? One possibility is using the 80C52 with its ROM BASIC interpreter. Few of the applications mentioned are really time-critical. However, assembly language should be supported for those hard-core fanatics (myself included) who think that using anything but assembler on a microcontroller is slothful and insipid.

As the Eliza program is wont to say, "Oh, I?"

The Obvious, Inc.

Every once in a while, I think that a new consulting company is needed. Call it "The Obvious, Inc." Every time a company comes out with something new or a new multimillion dollar advertising campaign, they could call in a consultant from The Obvious, Inc., to tell them the obvious, and thus prevent them from making really stupid moves.

For example, when Inmos advertises that the Transputer will become ubiquitous in waffle irons and microwave ovens: "It costs too much."

When the first NeXT [sic] came out with optical disks and a monochrome monitor: "The disks cost too much, the monochrome screen looks dull, and it's all black and photographs awful." The new version has a hard disk and a color screen, but it's still all black. Maybe it'll sell well in the funeral parlor market.

As for the IBM PS/1: "It's too expensive, and hard to expand." IBM, however, is wisely cautious; people are not junking their 286's and rushing to buy new 386 machines in spite of all of Intel's and Microsoft's pronouncements.

The reason that The Obvious, Inc. has to be a consulting company is that in-house people either don't, won't or can't see the obvious, or if they do, they aren't believed. As they say, "an Expert is someone from out of town."

Next time

Next time I hope to have some preliminary results from my PC-532. And, as always, no custom coffee mugs or T-shirts.•

Where to write or call

Minix:

Prentice-Hall 1-800-624-0023 or 1-201-767-5969
Microservice Customer Service
Simon & Schuster
200 Old Tappan Road
Old Tappan, NJ 07675

X-10:

Heathkit 1-800-253-0570
Heath Company
P.O. Box 8589
Benton Harbor, MI 49022
BBS: 1-703-330-9049

Forth and Forth Assembler in Embedded Systems

By Matthew Mercaldo

Why Forth?

The embedded systems field is the most dynamic of the engineering disciplines. Not only is chip technology advancing at a mind boggling rate, but firmware design methodology is following in close pursuit. The processors coming out now are accelerating; the applications are requiring more complex paradigms to complete the job. The engineer is required to keep up. Within the embedded systems world Forth, with an assembler extension, provides a superior approach to firmware design as well as instructional methodology. Forth allows the expression required to keep in touch with hybrid hardware technologies. *Forth provides a framework to express extremely complex software models in a concise form.*

Simplify the Development Cycle.

The firmware engineer can quickly verify microprocessor peripheral hardware in a Forth environment. In a traditional environment emulators are used to look at memory for vital clues as to the behavior of the hardware. Usually when the hardware's behavior is ambiguous, new code must be written, assembled, and burned into PROM, assembler listing generated with all the correct addresses and offsets, breakpoints set, etc.. This approach is costly. Forth affords the engineer a more complete view into his or her hybrid hardware, in any way he or she wishes to devise. Since words and assembler definitions can be compiled into into the Forth system interactively, extending the system, the engineer can look into his or her system under controlled varying conditions. Assembler words can be written, then referenced from hardware interrupt vectors, all of which can be done interactively. The engineer feels closer to the "metal" in Forth. *Forth gives the engineer the ability to verify custom hardware and determine it's actual behavior from the predicted behavior more quickly than the more traditional approach.*

Break Big Problems into Little Ones

In any software environment one is taught that the key to reliable software is the breaking down of the complex problem into simpler problems. This is the concept of factoring. The breaking down of problems becomes critical in a multitasking or multiple interrupt embedded system. In these systems there is a lot of asynchronisity. If the behavior of one of the components is not understood well enough, side effects become difficult to trace. In the traditional approach, a prom is burned or an intel hex image is downloaded, and away the program runs interrupts and all. Again, a costly emulator is required. In any scenario it can be difficult to evaluate interrupt driver code, especially if there are any algorithmic–time dependent portions. In these instances, Forth assembler allows the engineer to write interrupt drivers interactively. In certain instances this allows a more focused test of algorithmic dependency. Pieces of the driver can be removed, or added, or the driver can be simplified to verify hardware and processor time constraints. In other instances, non time-dependent portions of code can be tested directly from the Forth interpreter. Data analysis or preparatory algorithms may be tested from the interpreter first. When proven, these pieces can be introduced into some more time critical segments of the system for further analysis. *Forth gives the engineer a platform from which to factor complex problems and verify custom subsystems, then tune these systems algorithmically and or parametrically more quickly than the traditional approach.*

Give the Student a Clear Example.

Forth assembler, once integrated into a teaching methodology, reveals a powerful approach to education. Forth assemblers make no pretense of complexity. When writing assembler in a Forth environment, the student sees all that makes up an assembler instruction, clearly factored into its base components. An example is given below:

```
Forth assembler (68HC11):          0 ,X  A  LDA
traditional assembler (68HC11):    LDAA 0,X
```

Both lay down the same machine code of $A600, but the components are much more visible in the Forth style assembler. The student is forced to make a distinction. With each assembler instruction the concepts are asserted once again of addressing modes and operation, not a hazing of the two. Upon the completion of an assembler course using the Forth style of assembler, a student has obtained a thorough understanding of assembler concepts. These concepts are quickly extended to other CPU types of varying architecture. Forth assembler also has an extremely powerful macro capability; the assembler is extensible. This extensiblilty typically manifests itself in the structuring of the branching opcodes within an instruction set.

Once again an example is fitting:

```
.... TSTA
       BGE $1
       LDAA 0,X
       BRA $2
$1   LDAA 2,X
$2   .....
```

Is an assembler fragment representing the more traditional way.

```
A TST  GE IF
    2 ,X  A  LDA
ELSE
    0 ,X  A  LDA
THEN
```

Is an assembler fragment representing the Forth approach. Both lay down the same instructions, but the Forth approach is clearer. We have been taught that the structured approach is the most maintainable; Forth allows this, even at the assembler level.

Forth assembler also allows looping structures as illustrated below:

```
BEGIN
     B DEC  NE WHILE
          ABX  0 ,X  A  LDA
NE UNTIL
```

A piece of code that represents the above fragment in traditional assembler is illustrated below:

```
$1    DECB
      BEQ   $2
      ABX
      LDAA 0,X
      BEQ     $1
$2    ...
```

The Forth assembler code reads the same as a similar loop in high level Forth but lays down code identical to that of the traditional assembler loop. This loop is explained in the following example:

```
BEGIN                  Begin the loop
    B DEC  NE WHILE    Decrement B .. While Register B
                       is not zero,
        ABX    0 ,X A LDA Add B to X, Load what is pointed
                                      to by X into A
NE UNTIL                Until A is not equal to Zero.
```

The mechanics of control structure definition can be easily understood. They each take less than one line of Forth code to define, within the assembler's innerworkings, when all the pieces are in place . Forth itself is a tremendously powerful teaching tool. Forth allows the explanation of complex software concepts, simply, without the extra baggage associated with the more traditional systems. When these concepts are understood, the haziness of the environments interwoven with traditional systems clear away; the concepts which the student now owns can clearly be identified and correctly acted upon. A student who has learned software concepts using Forth has an easier time adapting to the dynamic world of engineering. The student who has learned software concepts with Forth owns those concepts. These give the student an ability to work in any software environment, using any of the typical engineering tools (soft or hard) with the highest degree of confidence. It is often said that a Forth trained programmer makes a better C programmer than a C trained programmer. This is not because of the language, but because of the intimate understanding of the fundamental concepts. Forth and Forth assembler give the student an opportunity to learn assembler concepts and fundamental software concepts more clearly; understanding the components which create things that the traditional approach typically hazes in cluttered environmental "baggage".

In conclusion.

As we have seen, Forth, with an assembler, affords the engineer an opportunity to understand the behavior of his or her hardware and software more intimately in a very cost effective environment. The student reaps the benefits of learning fundamental concepts that he or she can apply to any traditional environment and has a better understanding of the capabilities of that environment. This deeper understanding of problem solving gives any engineer or student an edge in a very challenging field.•

Matthew advises us that the F68HC11 Max-Forth assembler may be purchased for $50. Download it to a NewMicos board and you have a powerful Infix assembler. He may be contacted through The Computer Journal.

Join the

Forth Interest Group

The Forth Interest Group (FIG) continues to be the best Forth resource.

•**Forth Publications,** FIG carries the largest selection of Forth literature found anywhere.
• **Disk Library,** "Contributions from the Forth Community", includes tutorials and tools.
• **Forth Dimensions,** our bi–monthly magazine is devoted exclusively to Forth.
• **Chapters** provide an opprotunity for local, face–to–face meetings with other Forth enthusiasts.
• **GEnie™ Roundtable** provides a central focus for technical discussions and includes an on–line library of over 700 downloadable files.
• **Annual FORML Conference,** held at Asilomar Conference Center, on the beach in Pacific Grove, California, during the Thanksgiving holiday weekend, provides an excellent opprotunity to participate in technical sessions and mingle with leading Forth experts in an informal setting.

FIG is a non–profit, membership organization of over 1700 members in 20 countries. Membership includes a subscription to *Forth Dimensions,* discounts on purchases of Forth literature and more. Annual dues are $30 for USA, $36 for Canada air mail and $42 for all other countries. To join or receive further information, write or call *Forth Interest Group,* P.O. Box 8231, San Jose, CA 95155. Telephone: (408) 277-0668 or Fax: (408) 286-8988.

* GEnie (General Electric Network for Information Exchange) is a trademark of General Electric Company

Modula 2 and the TCAP

Writing Terminal Specific Code in Modula 2

By David L. Clarke

Introduction

In this article I will discuss how to write what I call cursor control code in Modula 2. This particular type of code will be slightly different for each terminal (I'll explain why). The most general solution is to use a mapping table to define the basic terminal operations. I will show how this is done with the Z-System TCAP. I will use this 'screens' module to graphically diagram the Z3 environment. For those who are not lucky enough to have a mapping table like the TCAP, I will show how a less general solution can be developed.

Terminal Specific Programming —the Problem

As you may have noticed from my 'bio', I work part time teaching Computer Science courses. One of my favorite classes discusses terminal programming. I cover a bit of history by showing how early terminals moved the cursor by control characters borrowed from the earlier Teletypes, specifically, backspace = left, (horizontal) tab = right, linefeed = down, and vertical tab = up. It soon became desirable to move the cursor by more than one position at a time. Therefore, each manufacturer came up with a series of characters that had special meaning, "move the cursor by more than one position." Later character sequences were developed to perform editing operations such as "delete rest of line" or "insert new stuff here." Since most manufactures decided to start these character sequences with the 'escape' character, they are often called "escape sequences." Unfortunately, the terminal manufacturers didn't work together too well at first. Each terminal had its own unique set of escape sequences. A 'standard' was eventually defined known as the ANSI standard (i.e. X3.64-1979), but the modern terminal maker is concerned with graphic capabilities that weren't even dreamt of when the ANSI standard was written. As such, the 'standard' is rather dated.

Now comes the practical part. I ask the class to imagine a computer facility that consists of a network of Sun workstations (like we have at the school). There are also several types of Perkin-Elmer terminals connected to the system (left over from an earlier computer). In addition to this there is a dial-in system where modem users (students) are able to call in and use the system. Some of these students may be using DEC computers with VT100 terminals, others may be using a Tektronix terminal, still others may be using a Commodore 128 or an Apple II. Each of these has its own set of escape sequences. "How," I ask, "would you write a full screen editor for this system that was able to handle all of these terminals and be expandable for future terminals as they are introduced?"

Some students feel that the solution

Listing 1.

```
DEFINITION MODULE ZScreens;

(*      D. L. Clarke            for TCJ         8 October 1990          *)

(*      Screen operation procedures using the Z-environment TCAP        *)

    PROCEDURE GoToXY(           (* move cursor to specific row and column *)
                col: CARDINAL;  (* the column       (x position)          *)
                row: CARDINAL); (* the row or line  (y position)          *)

    PROCEDURE HomeCursor;       (* move cursor to row 0, column 0         *)

    PROCEDURE ClearScreen;      (* Home cursor and clear screen           *)

    PROCEDURE ClearToEOL;       (* Clear from cursor to end of line       *)

    PROCEDURE ClearToEOS;       (* Clear from cursor to end of screen     *)

    PROCEDURE Highlight;        (* Enter highlight mode                   *)

    PROCEDURE Normal;           (* Exit Highlight mode                    *)

    PROCEDURE InitScreen;       (* Perform screen initial. sequence       *)

    PROCEDURE ExitScreen;       (* Perform screen de-initial. sequence    *)

END ZScreens.
```

David Clarke was originally an Electrical Engineer at Pratt & Whitney Aircraft until he discovered that it was more fun to program the data acquisition systems that he developed. He therefore became a systems programmer.

Dave is also an Adjunct Assistant Professor at the Hartford Graduate Center in Hartford CT., where he has taught courses in Systems Programming, Software Engineering, and Real Time Programming. Dave can be reached at the Graduate Center where his electronic mail address (Internet) is davec@mstr.hgc.edu. His home address for regular mail is P.O. Box 328, Tolland, CT. 06084.

is to require all terminals to conform to the ANSI standard. Those of us still using older terminals (or computers that are only able to emulate the older terminals) will realize that this solution is hard to enforce. Other students recommend a 'definition module' that defines the basic cursor motions. They would then create a different implementation module for each terminal type. The problem here is that, as a new terminal type logs in, either the editor program must be re-compiled with the right implementation module and linked, or else a separate executable must be kept for each terminal type.

A better solution is suggested by taking a closer look at the school's system. After all, it was the model for the question. The school system runs under the UNIX operating system. UNIX maintains a table that contains entries for hundreds of terminal types. Each table entry maps codes for the basic cursor motions into specific escape sequences. As a student logs in, the appropriate terminal entry is extracted from the table and becomes part of the user's local environment. The editor program need only examine the local codes to determine how to perform the desired operations. UNIX calls this table the TERMCAP which stands for TERminal CAPabilities.

The Z-System TCAP

The ZCPR3 environment contains a similar structure called the TCAP. The Z-System user is able to select a specific terminal definition from a library of definitions. This definition is first loaded into the Z environment. Programs are then able to examine the environment to determine how to perform the basic cursor movements.

At this time we should take a quick look at the Z3 TCAP. A Modula 2 'RECORD' for the TCAP was at the beginning of the Z34M2 definition module in my last article. However, most of the important part of it is hidden in a single ARRAY OF CHAR. This is because most of these 'hidden' sequences are variable length strings terminated by a zero byte. We will eventually look at these strings individually.

The first TCAP field consists of a 15 character terminal name. The next byte contains bits that describe such terminal modes as whether highlighting is available as well as line wrapping and scrolling characteristics. The next four bytes

Listing 2.

```
IMPLEMENTATION MODULE ZScreens;

(*      D. L. Clarke            for TCJ              8 October 1990      *)

FROM Terminal IMPORT Write, WriteString, WriteLn;
FROM SYSTEM   IMPORT ADDRESS, ADR;
FROM z34m2    IMPORT TCAP, Z3PTR, GetEnv;

TYPE    String = ARRAY [0 .. 14] OF CHAR;
        string = POINTER TO String;
        pTCAP  = POINTER TO TCAP;

VAR     i:      CARDINAL;
        env:    Z3PTR;
        tcap:   pTCAP;          aTCAP:  ADDRESS;
        al:     string;         AL:     ADDRESS;    (* add line *)
        cd:     string;         CD:     ADDRESS;    (* clear to EOS *)
        ce:     string;         CE:     ADDRESS;    (* clear to EOL *)
        cl:     string;         CL:     ADDRESS;    (* clear screen *)
        cm:     string;         CM:     ADDRESS;    (* cursor motion *)
        dl:     string;         DL:     ADDRESS;    (* delete line *)
        se:     string;         SE:     ADDRESS;    (* end standout *)
        so:     string;         SO:     ADDRESS;    (* start standout *)
        te:     string;         TE:     ADDRESS;    (* terminal de-init *)
        ti:     string;         TI:     ADDRESS;    (* terminal init *)

    PROCEDURE GoToXY(col, row: CARDINAL);
        VAR     i, val: CARDINAL;
                first:  BOOLEAN;
                ch:     CHAR;
    BEGIN
        i := 0;             first := TRUE;
        LOOP
            IF (i > 15) OR (cm^[i] = 0c) THEN EXIT END;
            IF cm^[i] = '%' THEN
                IF first THEN val := row;    first := FALSE
                         ELSE val := col
                END;
                INC(i);
                CASE CAP(cm^[i]) OF
                    'I':            INC(row);    INC(col);
                                    first := TRUE
                |   'R':            row := col;  col := val;
                                    first := TRUE
                |   '+':            INC(i);
                                    ch := CHR(ORD(cm^[i]) + val);
                                    Write(ch)
                |   'D', '2', '3': ch := CHR(val DIV 100 + ORD('0'));
                                    IF ((CAP(cm^[i]) = 'D') AND (ch # '0'))
                                       OR (cm^[i] = '3') THEN
                                        Write(ch)
                                    END;
                                    ch := CHR(val MOD 100 DIV 10 + ORD('0'));
                                    IF (cm^[i] = '2') OR (cm^[i] = '3')
                                       OR ((CAP(cm^[i]) = 'D') AND (ch # '0'))
                                       OR (val >= 100) THEN
                                        Write(ch)
                                    END;
                                    ch := CHR(val MOD 10 + ORD('0'));
                                    Write(ch)
                    ELSE            Write(cm^[i])
                END  (* case *)
            ELSE
                Write(cm^[i])
            END;
            INC(i)
        END   (* loop *)
    END  GoToXY;

    PROCEDURE HomeCursor;
    BEGIN    GoToXY(0, 0)        END  HomeCursor;

    PROCEDURE ClearScreen;
    BEGIN    WriteString(cl^)    END  ClearScreen;

    PROCEDURE ClearToEOL;
    BEGIN    WriteString(ce^)    END  ClearToEOL;
```

contain the codes that are generated by the cursor arrow keys if they exist, otherwise you get to select your own codes. Usually these bytes are ASCII control characters.

Some of the older terminals required quite a lot of time to perform some functions. The next three bytes in the TCAP contain the delay times for clearing the screen, moving the cursor, and clearing to the end of the current line.

The remaining codes usually take more than a single byte. They are the escape sequences used by the terminal to accomplish the desired functions. Each escape sequence is terminated by a byte containing a value of zero (i.e., the NUL character). Some terminals may use a single control character to perform a given function, but the string will take up two bytes because of the NUL. If a terminal cannot perform a particular function, the corresponding string will consist of a single NUL byte. The strings in the TCAP define the following functions in the order listed:

```
Clear the entire screen
Move the cursor to a particular row and column
Clear to the end of the current line
Begin Highlight mode (e.g. brighter or reverse video)
End highlight mode
Initialize terminal (if required)
De-initialize terminal (return to original state)
Delete line (next line moves up to take its place)
Insert (blank) line before the current line
Clear to the end of the screen
```

The latest version of the TCAP definition also contains a series of graphics definitions that will not be discussed here.

The code to move the cursor to a particular row and column is quite interesting. It is this string that shows us how much was borrowed from the UNIX TERMCAP. The string uses the percent character, '%', to indicate several things. Usually '%' indicates where the value for the row or column should be substituted into the string. (Normally the row is the first '%' indicated.) It also indicates how the value is to be inserted into the string. For instance, if the character following the '%' is a 'd', a string of decimal digits is to be inserted. If instead of the 'd', there is a '2' or '3', then the string of digits must be exactly 2 or 3 digits long respectively and use preceding zeros if necessary. A string in the form '%+c' indicates that the ascii value of the character, 'c', is to be added to the coordinate value. The result is the ASCII value of the character that gets inserted into the string for this coordinate. The '%' character is also used for two other purposes. Normally the upper left corner of the screen is considered to be row 0 and column 0, i.e. (0, 0). A '%i' code says to increment the values passed to it before encoding them and inserting them into the string. Thus the upper left corner encodes as (1, 1). A '%r' code says to reverse the row/column order. Thus column becomes the first value to be encoded. For example row 10, column 20 becomes:

```
<esc> [21;11H   with a code of
                1Bh,'[%i%r%d;%dH'
                -or-
=*4             with a code of
                1Bh,'=%+ %+ '
```

Where <esc> and 1Bh are the escape character.

Listing 2. Continued

```
    PROCEDURE ClearToEOS;
    BEGIN    WriteString(cd^)    END  ClearToEOS;

    PROCEDURE Highlight;
    BEGIN    WriteString(so^)    END  Highlight;

    PROCEDURE Normal;
    BEGIN    WriteString(se^)    END  Normal;

    PROCEDURE InitScreen;
    BEGIN    WriteString(ti^)    END  InitScreen;

    PROCEDURE ExitScreen;
    BEGIN    WriteString(te^)    END  ExitScreen;

BEGIN      (*  Initialisation  *)
    env := GetEnv();
    IF CARDINAL(env) # 0 THEN
        aTCAP := ADDRESS(env);      INC(aTCAP, 128);     tcap := pTCAP(aTCAP);
        cl := ADR(tcap^.CtlStr);
        i := 0;    WHILE cl^[i] # 0c DO INC(i) END;
        CM := ADR(cl^[i]);          INC(CM, 1);          cm := string(CM);
        i := 0;    WHILE cm^[i] # 0c DO INC(i) END;
        CE := ADR(cm^[i]);          INC(CE, 1);          ce := string(CE);
        i := 0;    WHILE ce^[i] # 0c DO INC(i) END;
        SO := ADR(ce^[i]);          INC(SO, 1);          so := string(SO);
        i := 0;    WHILE so^[i] # 0c DO INC(i) END;
        SE := ADR(so^[i]);          INC(SE, 1);          se := string(SE);
        i := 0;    WHILE se^[i] # 0c DO INC(i) END;
        TI := ADR(se^[i]);          INC(TI, 1);          ti := string(TI);
        i := 0;    WHILE ti^[i] # 0c DO INC(i) END;
        TE := ADR(ti^[i]);          INC(TE, 1);          te := string(TE);
        i := 0;    WHILE te^[i] # 0c DO INC(i) END;
        DL := ADR(te^[i]);          INC(DL, 1);          dl := string(DL);
        i := 0;    WHILE dl^[i] # 0c DO INC(i) END;
        AL := ADR(dl^[i]);          INC(AL, 1);          al := string(AL);
        i := 0;    WHILE al^[i] # 0c DO INC(i) END;
        CD := ADR(al^[i]);          INC(CD, 1);          cd := string(CD);
    ELSE
        WriteString("No ZCPR3.3+ environment"); WriteLn
    END
END ZScreens.
```

The ZScreens Module

I developed the ZScreens module to allow Modula 2 to access the TCAP environment. Listing 1 shows the definition module for ZScreens. It contains most of the normal cursor motion and editing functions. Procedure 'GoToXY' will position the cursor anywhere on the screen. 'HomeCursor' moves the cursor into the upper left-hand corner of the screen (i.e. the 'home' position). 'ClearScreen ' will clear the complete screen and leave the cursor in the home position. (Or does it home the cursor and then clear to the end of the screen?) 'ClearToEOL' clears from the cursor to the end of the current line. 'ClearToEOS' clears from the cursor to the end of the screen. 'Highlight' will put the terminal in highlight mode, which could be brighter or dimmer than the normal mode, or it might be reverse video—it all depends on how it's defined in the TCAP. The 'Normal' procedure returns the terminal to the normal (un-highlighted) mode. The 'InitScreen' procedure will output the terminal initialization character sequence. Likewise, 'ExitScreen' will output the terminal de-initialization char-

acter sequence.

Listing 2 shows the ZScreens implementation module. Most of the work of the module is done in the initialization section. This means that it is actually done before the main code of the program begins to run. With the exception of GoToXY, all of the procedures in the module simply write a string directly from the TCAP. GoToXY is more complex because it must interpret the '%' codes in the string.

The implementation begins by getting the address of the Z3 environment. The TCAP is always located 128 bytes above this address. This module is not concerned with the first few fields of the TCAP RECORD (as defined in my previous article). Instead we start at the 'CtlStr' ARRAY OF CHAR. The first sequence in this array is the 'clear the entire screen' string which is often referred to by the mnemonic 'cl'. In this module, 'cl' is set to point to the beginning of CtlStr. Then a search is made down the string for the NUL character. The next higher location is the address of the beginning of the 'move the cursor' string. The 'cm' variable is made to point to this location. Again, a search is made for the next NUL character. The following location begins the 'clear to EOL' string. In like manner, all of the control sequences are found and have a variable assigned to them that contains the address of their string in the TCAP. Thus, when it comes to implementing one of the procedures such as 'ClearScreen', all the procedure has to do is write the string pointed to by the appropriate variable (in this case, 'cl'). Of course, it helps that Modula 2 also uses the NUL character to indicate the end of a string.

The 'move cursor' string is pointed to by the variable 'cm'. The GoToXY procedure cannot write the string directly. Instead, it writes the string out one character at a time until a '%' is detected. The '%' itself is not output, however the next character is used to determine how to encode the row and column values. The encoding is performed according to the rules given earlier. In this manner, the correct character sequence is generated on the fly.

Mapping the Environment

In my last article, I showed how some of the information in the Z-System environment could be listed by a modula 2 program. Now I shall show how the actual environment can be mapped graphically using the Screens module. The Z3 environment contains pointers to its component parts. A glance at the Z3ENV record from the

Listing 3.

```
MODULE MapEnv;

(*      D. L. Clarke            for TCJ             8 October 1990        *)

FROM SmallIO  IMPORT WriteHex;
FROM SYSTEM   IMPORT ADDRESS;
FROM Terminal IMPORT Write, WriteString, WriteLn;
FROM z34m2    IMPORT Z3ENV, Z3PTR, GetEnv;
FROM ZScreens IMPORT Highlight, Normal, ClearScreen, GoToXY;

VAR     location, page: INTEGER;
        line:           CARDINAL;
        x, y, addr:     CARDINAL;
        env:            Z3PTR;

PROCEDURE FindXY(location: CARDINAL);   (* find map location of structure *)
    BEGIN
        y := 65 - location DIV 1024;
        x := location MOD 1024 DIV 32 + 10
    END  FindXY;

PROCEDURE PlaceName(locat: ADDRESS;     (* place name on map and in list  *)
                    name: ARRAY OF CHAR;
                    size: CARDINAL);
    BEGIN
        addr := CARDINAL(locat);
        IF (addr >= 0b000H) AND (size >= 128) THEN  (* place name on diagram *)
            FindXY(addr);     GoToXY(x, y);
            Highlight;        WriteString(name);       Normal
        END;
        INC(line);            GoToXY(48, line);      (* list beside diagram *)
        Highlight;            WriteString(name);       Normal;
        GoToXY(55, line);     WriteHex(addr, 4);
        GoToXY(62, line);     WriteHex(size, 4)
    END   PlaceName;

BEGIN
    env := GetEnv();
    IF CARDINAL(env) # 0 THEN
        ClearScreen;
        GoToXY(18, 0);      Highlight;
        WriteString("> > >  ZCPR Environment Map  < < <");
        Normal;             WriteLn;
        WriteString("         +");
        FOR page := 0 TO 3 DO           (* draw the top part of the diagram *)
            WriteString("-");
            Highlight;      WriteHex(page*100H, 3);    Normal;
            WriteString("-+")
        END;    WriteLn;
        FOR location := -1024 TO -20480 BY -1024 DO     (* draw the diagram *)
            WriteString("   ");    WriteHex(location, 4);   WriteString("  |");
            FOR page := 0 TO 3 DO  WriteString("       |")  END;
            WriteLn
        END;
        line := 4;
        GoToXY(48, line);  WriteString("Name___Addr___Size");
        PlaceName(env^.bios,   "BIOS", 0ffffH - CARDINAL(env^.bios) + 1);
        PlaceName(env^.ccp,    "CCP",  ORD(env^.ccps) * 128);
        PlaceName(env^.dos,    "DOS",  ORD(env^.doss) * 128);
        PlaceName(env^.z3env,  "ENV",  ORD(env^.z3envs) * 128);
        PlaceName(env^.fcp,    "FCP",  ORD(env^.fcps) * 128);
        PlaceName(env^.iop,    "IOP",  ORD(env^.iops) * 128);
        PlaceName(env^.z3cl,   "MCL",  ORD(env^.z3cls));
        PlaceName(env^.z3msg,  "MSG",  80);
        PlaceName(env^.z3ndir, "NDR",  ORD(env^.z3ndirs) * 18 + 1);
        PlaceName(env^.expath, "PATH", ORD(env^.expaths) * 2 + 1);
        PlaceName(env^.rcp,    "RCP",  ORD(env^.rcps) * 128);
        PlaceName(env^.shstk,  "SHL",  ORD(env^.shstks) * ORD(env^.shsize));
        PlaceName(env^.extstk, "STK",  48);
        PlaceName(env^.z3whl,  "WHL",  1);
        PlaceName(env^.extfcb, "XFCB", 36);
        GoToXY(72, 22)
    END
END   MapEnv.
```

Listing 4.

```
>>> ZCPR Environment Map <<<

      000   |  100  |  200  |  300  |

FC00        |       |       |       |
F800        |       |       |       |
F400        |       |       |       |
F000  BIOS  |       |       |       |
EC00        |       |       |       |
E800  SHL   |       |  DOS  |       |
E400  FCP   |       |       |  ENV  |
E000        |       |       |       |
DC00  NDR   |  RCP  |       |       |
D800        |       |       |       |
D400        |       |       |       |
D000        |  MCL  |  CCP  |       |
CC00        |       |       |       |
C800        |       |       |       |
C400        |       |       |       |
C000        |       |       |       |
BC00        |       |       |       |
B800        |       |       |       |
B400        |       |       |       |
B000        |       |       |       |

Name    Addr    Size
BIOS    F000    1000
CCP     D200    0A00
DOS     EA00    0600
ENV     E780    0100
FCP     E500    0280
IOP     0000    0000
MCL     D101    00FA
MSG     E900    0050
NDR     DC00    00FD
PATH    E980    000B
RCP     DD00    0800
SHL     E880    0080
STK     E9A5    0030
WHL     E98B    0001
XFCB    E950    0024
```

z34m2 definition module listed in my last article will show that addresses are given for the RCP, IOP, etc., as well as the CCP, DOS, and the BIOS. The trick is to paint it on the terminal screen.

I chose to map only the portion of memory above hex location B000. Each line of the screen will represent 400 hex bytes (or 1024 decimal bytes). I then chose to let each character-space equal 32 bytes. Thus, 32 characters would be sufficient to represent the 1024 bytes assigned to the line. A vertical line, '|', every eight spaces will partition the line into areas of 100 hex bytes each. This is the matrix into which the environment shall be mapped.

Listing 3 shows the MapEnv program. It first gets the address of the environment. The address is valid if the address is non-zero. If it's valid we proceed with the mapping. The next fifteen or so lines are the code that draws the matrix. After this is a series of calls to 'PlaceName' for all of the interesting environmental elements.

Procedure 'PlaceName' only lists elements on the matrix that are at least 128 bytes long (i.e. four characters—the length of most names). If it's big enough, a call is made to 'FindXY' which calculates where the name belongs in the matrix. Then the Screens procedure 'GoToXY' is called to position the cursor there and the element's name is printed. In addition, the names and sizes of all of the elements is listed beside the matrix. The final result will look something like Listing 4.

The VTScreens Module

Sometimes a general solution (such as using the TCAP) is not possible. In a situation where there is only one terminal type to consider, a more specific approach may be taken. This might be the case with any system that has only one terminal type, such as an IBM PC. I will now show an example of this sort. I have chosen to implement a screens module for the VT100. This is the terminal most people associate with the ANSI standard, although there are others. In addition, it is possible to instruct the IBM PC to emulate a subset of the ANSI standard. Therefore this module should be usable in many situations.

To simplify things, and to retain some commonality, we shall use the same definition module as before, just change all references from "ZScreens" to "VTScreens." Next we get a copy of the ANSI standard and translate it into an implementation module. Listing 5 shows the final result. If you are using a VT100, you're ready, just link this module into any other programs and use it like ZScreens was used. If you are using an IBM PC, you will have to insert the following line into your CONFIG.SYS file:

```
device = [d:][path]ansi.sys
```

where 'd:' and 'path' may be optional. This will increase the size of DOS in memory, but you will be (almost) ANSI compatible.

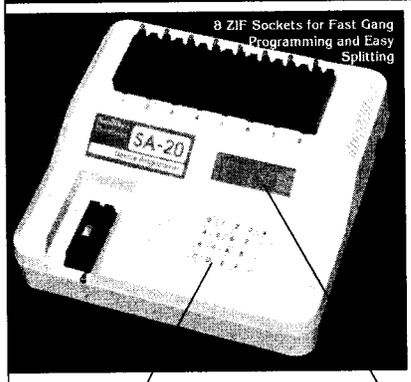
Conclusion

Two Screens modules were developed in this article. A handy program was included that used the ZScreens module to map the Z3 environment. This is only the beginning of what can be done with these modules. For instance, the calculator from my first article would seem a lot 'cleaner' if it positioned all answers into a single window on the screen instead of letting all of the answers scroll up and off the screen.

I hope that people are beginning to appreciate what can be done with Modula 2. I find that I can usually do things with it that previously I had to write in assembler language. Next time I hope to do some more work with the Z-System environment.•

EPROM PROGRAMMERS

Stand-Alone Gang Programmer **$750.00**

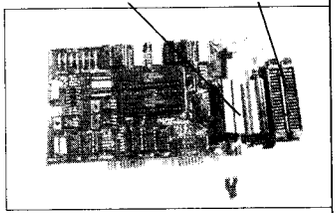

8 ZIF Sockets for Fast Gang Programming and Easy Splitting
SA-20
20 Key Tactile Keypad (not membrane)
20 x 4 Line LCD Display

- Completely stand-alone or PC driven
- Programs E(E)PROMs
- ***1 Megabit of DRAM***
- ***User upgradable to 32 Megabit***
- ***.3/.6" ZIF socket, RS-232, Parallel In and Out***
- 32K internal Flash EEPROM for easy firmware upgrades
- ***Quick Pulse Algorithm (27256 in 5 sec, 1 Megabit in 17 sec.)***
- 2 year warranty
- Made in U.S.A.
- Technical support by phone
- Complete manual and schematic
- ***Single Socket Programmer also available. $550.00***
- Split and Shuffle 16 & 32 bit
- 100 User Definable Macros, 10 User Definable Configurations
- Intelligent Identifier
- Binary, Intel Hex, and Motorola S

Internal Programmer for PC **$139.95**

New Intelligent Averaging Algorithm. Programs 64A in 10 sec., 256 in 1 min., 1 Meg (27010, 011) in 2 min. 45 sec., 2 Meg (27C2001) in 5 min. Internal card with external 40 pin ZIF.

2 ft. Cable 40 pin ZIF



- Reads, verifies, and programs 2716, 32, 32A, 64, 64A, 128, 128A, 256, 512, 513, 010, 011, 301, 27C2001, MCM 68764, 2532
- ***Automatically sets programming voltage***
- Load and save buffer to disk
- Binary, Intel Hex, and Motorola S formats
- ***Upgradable to 32 Meg EPROMs***
- ***No personality modules required***
- 1 year warranty • 10 day money back guarantee
- Adapters available for 8748, 49, 51, 751, 52, 55, TMS 7742, 27210, 57C1024, and memory cards
- Made in U.S.A.

NEEDHAM'S ELECTRONICS

4539 Orange Grove Ave. • Sacramento, CA 95841
Mon. - Fri. 8am - 5pm PST

Call for more information
(916) 924-8037
FAX (916) 972-9960

C.O.D.

Listing 5.

```
IMPLEMENTATION  MODULE  VTScreens;

(*      D. L. Clarke            for TCJ         10 October 1990         *)

(*      This module is written specifically for the VT-100. It is hoped *)
(*   that this will also work with the Sun terminal and the IBM PC in   *)
(*   ANSI mode (however, refer to ClearToEOS).                          *)

  FROM  Terminal  IMPORT  Write, WriteString;
  FROM  SmallIO   IMPORT  WriteCard;

  CONST esc = 033c;

  PROCEDURE HomeCursor;
  BEGIN
    GoToXY(0, 0)
  END  HomeCursor;

  PROCEDURE ClearScreen;
  BEGIN
        Write(esc);         WriteString("[2J")
  END  ClearScreen;

  PROCEDURE ClearToEOL;
  BEGIN
        Write(esc);         WriteString("[K")
  END  ClearToEOL;

  PROCEDURE ClearToEOS;     (* Note - this may not work with IBM PC *)
  BEGIN
        Write(esc);         WriteString("[J")
  END  ClearToEOS;

  PROCEDURE Highlight;
  BEGIN
        Write(esc);         WriteString('[1m');   (* enter BOLD mode *)
  END Highlight;

  PROCEDURE Normal;
  BEGIN
        Write(esc);         WriteString('[0m');   (* exit BOLD mode *)
  END Normal;

  PROCEDURE InitScreen;
  BEGIN
        Write(esc);         Write('c')      (* reset to initial state *)
  END InitScreen;

  PROCEDURE ExitScreen;
  BEGIN
        Write(esc);         Write('c')      (* reset to initial state *)
  END ExitScreen;

  PROCEDURE GoToXY(x, y: CARDINAL);
  BEGIN
        Write(esc);         Write('[');
        WriteCard(y+1, 1);  Write(';');
        WriteCard(x+1, 1);  Write('H')
  END  GoToXY;

END VTScreens.
```

Call for Articles

The Computer Journal *relies on its readership for submission of articles. Interest has been expressed for articles relating to real-world interfacing and programming of embedded controllers and robotics, internal structure and modification of MS-DOS and Z80 assembly programming for the high level language programmer as it relates to Z-System. Articles should be submitted in ASCII format and be between 10k and 30k in length. Graphics are accepted in TIFF and EPS formats. Please send submissions to:*

The Computer Journal
PO Box 12
South Plainfield, NJ 07080-0012
United States

or by modem to Socrates Z-Node 32, (908) 754-9067

8031 µController Modules

NEW!!!

Control–R II

√ Industry Standard 8–bit 8031 CPU
√ 128 bytes RAM / 8 K of EPROM
√ Socket for 8 Kbytes of Static RAM
√ 11.0592 MHz Operation
√ 14/16 bits of parallel I/O plus access to address, data and control signals on standard headers.
√ MAX232 Serial I/O (optional)
√ +5 volt single supply operation
√ Compact 3.50" x 4.5" size
√ Assembled & Tested, not a kit

$64.95 each

Control–R I

√ Industry Standard 8–bit 8031 CPU
√ 128 bytes RAM / 8K EPROM
√ 11.0592 MHz Operation
√ 14/16 bits of parallel I/O
√ MAX232 Serial I/O (optional)
√ +5 volt single supply operation
√ Compact 2.75" x 4.00" size
√ Assembled & Tested, not a kit

$39.95 each

Options:
• MAX232 I.C. ($6.95ea.)
• 6264 8K SRAM ($10.00ea.)

Development Software:
• PseudoSam 51 Software ($50.00) Level II MSDOS cross–assembler. Assemble 8031 code with a PC.

• PseudoMax 51 Software ($100.00) MSDOS cross–simulator. Test and debug 8031 code on your PC!

Ordering Information:
Check or Money Orders accepted. All orders add $3.00 S&H in Continental US or $6.00 for Alaska, Hawaii and Canada. Illinois residents must add 6.25% tax.

Cottage Resources Corporation
Suite 3–672, 1405 Stevenson Drive
Springfield, Illinois 62703
(217) 529-7679

The Z-System Corner

Patching MEX-Plus and The Word – Using ZEX

By Jay Sage

For this issue I have a large number of small subjects to cover. I will begin with the long-promised patches to MEX-Plus, the ones I lost when my hard disk crashed and then recovered when the hard disk miraculously resurrected itself. I haven't had time to incorporate all the corrections and additions that have been on my list, but this will be a good start on an on-going project.

While on the subject of patching, I will enlarge on the discussion of the patch to The Word Plus that I presented in issue #45. It turns out that at least two different versions of TW.COM are currently in circulation, and the original patch works only with one.

A couple of readers have requested help with ZEX. They are aware that Bridger Mitchell provided full documentation in his column in issue #38, but they pointed out that the subject is rather complex and would be clarified considerably by a few examples. Given the power of command alias scripts, I rarely have any need for ZEX, but I will present the two ZEX scripts that I do use regularly. Perhaps some readers will help out by sending me copies of some interesting scripts that they have developed that use the advanced features of ZEX5.

Finally, I will continue my discussion of ZMATE. Rather than listing the remaining commands that I did not cover last time, I will describe my special autoexec macro that allows me to pass macro commands to ZMATE from the operating system command line. This provides a very powerful interface between ZMATE and Z-System command alias scripts.

This is not the only information about ZMATE in this issue. I am happy to announce that we are inaugurating a new column by Clif Kinne on MATE macros.

One other quick announcement. The *Z-System Software Update Service*, produced by Bill Tishey and Chris McEwen, is expanding its services. ZSUS will now make up custom disks containing the full LBR releases of any files you request. There will also soon be special software collections. I believe the first will contain material of interest to assembly language programmers. I invite you to write to me for a new flyer or to look for announcements on BBSs.

Fixing Up MEX-Plus

As wonderful as I find MEX-Plus, there are quite a few bugs that need to be fixed and quite a few additional features that I would like to see added. I have just gotten started on this project and will describe my progress to date.

MEX Improvement List

Besides the items that my current patch deals with, the following items are on my list of needed improvements.

The CLONE command, which creates a new copy of MEX (i.e., a new COM file) with the current settings as the default, has the following minor bug: when it is entered without specifying a file name, it will make a file with extent "COM" and a file name of all spaces! In this case, it would be much more sensible if it either (1) defaulted to a name of MEX.COM or (2) refused to do anything.

There is a very annoying error in the way numerical variables are processed. When the variables have particular values, MEX reports a syntax error. So far I have not been able to determine the precise conditions under which the problem arises, but I hope that with careful study of the code I will figure out what is wrong. My guess is that some CPU flag is being mis-set or misinterpreted.

There is a minor bug in the DATE command. As things are now, TIME and DATE are identical functions, and both return the time in the VALUE variable. Obviously, DATE should return the current date.

In case you have tried to use the MENU.MOD module and have found that the HELP function does not work, don't worry. You are not doing anything wrong or missing any piece of the code. This function was implemented improperly or not implemented at all in the CP/M version of MEX-Plus. We probably will not be able to do much about this until we have figured out MEX's interface to the external modules.

Jay Sage has been an avid ZCPR proponent since the very first version appeared. He is best known as the author of the latest versions 3.3 and 3.4 of the ZCPR command processor, his ARUNZ alias processor and ZFILER, a "point-and-shoot" shell.

When Echelon announced its plan to set up a network of remote access computer systems to support ZCPR3, Jay volunteered immediately. He has been running Z-Node #3 for more than five years and can be reached there electronically at 617-965-7259 (MABOS on PC Pursuit, 8796 on Starlink, pw=DDT). He can also be reached by voice at 617-965-3552 (between 11 p.m. and midnight is a good time to find him at home) or by mail at 1435 Centre Street, Newton Centre, MA 02159. Jay is now the Z-System sysop for the GEnie CP/M Roundtable and can be contacted as JAY.SAGE via GEnie mail, or chatted with live at the Wednesday real-time conferences (10 p.m. Eastern time).

In real life, Jay is a physicist at MIT, where is tries to invent devices and circuits that use analog computation to solve problems in signal, image and information processing. His recent interests include artificial neural networks and superconducting electronics. He can be reached at work via Internet as SAGE@LL.LL.MIT.EDU.

I would like to mention something that caused me grief recently with one of my macros: WAIT STRING strings are limited to 16 characters! I don't think I would change this, but one must bear this limitation in mind.

One of the most important additions would be more string variables beyond the six (A..F) currently implemented. Even two more would benefit some of my scripts enormously.

Finally, I would like to replace the file name parser, which already recognizes the DU: prefix, with one that handles full Z-System file specifications.

My MEX Patch

My MEX patch is shown in Listing 1. It is heavily commented, and I will not repeat the details in the text. The patch makes three changes.

MEX has a STAT command that turns on or off a function that converts the backspace key into a rubout/delete character (which we will call DEL from now on). I have always assumed that this was included from olden times, when microcomputers sometimes had no DEL key and mainframe computers—which MEX was used to communicate with—required that key for some critical functions.

Today, the situation is generally quite different. First, almost all computers and terminals now have a DEL key. Second, with the Z-System, one often uses DEL interchangeably with backspace, and I have gotten quite used to that. When I call non-Z remote access systems, I find it a real nuisance when DEL does not work. That is why I decided to reverse the function of this STAT command.

Making the change itself was quite simple; the hard part was finding the right places in the code. Three changes are required, as shown at the end of Listing 1. One byte contains the keycode of the key to be intercepted, while another byte contains the keycode with which it is to be replaced. The third change is not needed for proper operation, but it is nice to have the STAT RUB message reflect the new function.

The next change is to the STAT BUF command. MEX very often has its commands return in the VALUE variable some information developed by that function. For example, the DIR function returns the number of matching files that were found. Unfortunately, the STAT BUF command displays information about the status of the capture and key macro buffers but does not return any information in VALUE.

I had written a script for calling up GEnie and capturing my email letters. When I relied on XON/XOFF flow control to pause GEnie's output while MEX was flushing its buffer, I found that some text was sometimes lost. To get around this, I manually closed and reopened the capture buffer. Rather than wasting the time to do this after each letter, I wanted to check the amount of space remaining in the capture buffer. If this space dropped below some value (I think I used 3K), then I would close and reopen the buffer.

Implementing this function required adding some new instructions to the code for the STAT BUF command. This raised two problems: how to tap into the original code and where to put the new instructions.

Let us start with the second problem. With some programs, new code can be added at the end of the original code. If the program places its run-time data at the end of its code, however, this can obviously present some difficulties. If one is patching in some new initialization code that is performed and can be discarded before the program starts to

Listing 1.

```
; Program:              MEXPAT.Z80
; Author:               Jay Sage
; Created:              June 10, 1990
; Last Modified:        October 8, 1990

; This file contains a collection of patches made to Mex-
; Plus, version 1.65.

; The following patches are included:
;
;    1. Change the function of the backspace-to-rubout
;       conversion function to do the reverse - make
;       pressing the rubout/delete key generate a backspace
;       character.  This patch could be used to perform the
;       conversion of any single character into any other
;       single character.  This function is invoked by the
;       STAT RUB [ON|OFF] command.
;
;    2. Augment the STAT BUFFER command so that the amount
;       of free space in the capture buffer is returned in
;       the VALUE variable.  MEX scripts can then check on
;       the remaining space and decide to flush the buffer
;       manually at convenient times and to avoid overflow
;       problems that can occur when relying on XON/XOFF
;       control.
;
;    3. Augment the BELL command so that pressing any key
;       will terminate it.  The standard MEX BELL command
;       can be cancelled only by control-C, but this
;       terminates the entire script.  Now the bell can
;       sound until the operator presses a non-abort key, at
;       which point the script will proceed.

; Addresses in standard code

ilprint     equ 0724fh          ; In-line print
hlsubde2    equ 0417ah          ; Subtraction of DE from HL
capstats    equ 06eafh          ; CAPSTATS function
scrollconin equ 049d3h          ; Console input
getconstat  equ 049c5h          ; Console status
bell2       equ 046c2h          ; BELL2 entry point

val     equ     00d64h          ; VALUE variable

logo    equ     02662h          ; LOGO entry point
endlogo equ     027b9h          ; First byte past LOGO code

saycap  equ     06ea6h          ; Point to patch in
                                ; ...SAYCAPSTATS
bellins equ     046aeh + 10     ; Point to patch in BELL

oldkey  equ     5351h           ; Place where key typed by
                                ; ...user is detected
newkey  equ     535dh           ; Place where new character
                                ; ...is substituted
idstr   equ     62a6h           ; Place where STAT command
                                ; ...message is defined

bs      equ      8              ; Backspace character
lf      equ      10             ; Linefeed
cr      equ      13             ; Carriage return
rubout  equ     127             ; Rubout character

; ---------------------------

; Standard MEX-Plus has a very elaborate signon screen that
; affords a perfect place to put the code we need for our
; patches.  The first thing we do is to replace the LOGO
; routine with a minimal message.

        org     logo

        call    ilprint
        db      cr,lf
        db      'MEX v 1.65Z, 06/10/90'
        db      cr,lf,lf,0
        ret

; Make the STAT BUFFER command put the amount of free
; capture buffer space into the VALUE variable.  We do this
```

use its data space, then one can get away with this. When I add this kind of patch, I try to have it include code that patches out the patch and restores the original code. Otherwise, the program cannot safely be rerun using the GO command, since the patch code may have been overwritten by program data. Attempting to execute data has a way of wreaking havoc with the system!

In our present case, we are adding code that must be available at any time, and MEX certainly writes data into the free memory after its own code. Consequently, we have to find some space within the bounds of the original program. The technique I have used here is to steal nonessential internal data space.

MEX normally signs on with a very elaborate screen display that takes 343 bytes—enough room for a lot of patch code! We cannot delete all of it, but we can replace it with a much simpler signon message. The new one in the patch uses only 31 bytes (if I counted correctly). It lets us know that we are running the special version of MEX and tells us the date of the patch.

Listing 1. Continued

```
; by patching some extra code into the SAYCAPSTATS routine.
cappat:
        call    capstats        ; Get status info on capture
                                ; ...buffer
        push    hl
        ld      h,b             ; Free space (BC) into HL
        ld      l,c
        ld      (val),hl        ; Put it into VALUE
        pop     hl
        jp      capcont

; This is a new routine that checks to see if ANY key has
; been pressed, not just control-c.  It will be used in the
; patched BELL routine.

chkchar:
        call    getconstat      ; See if key pressed
        ret     z               ; Return now if not
flush:
        call    scrollconin     ; Read in the character
        or      a               ; Set flag to nonzero
        ret

; ---------------------------

; Code for cutting into the original code in CAPSTATS

        org     saycap
        jp      cappat
capcont:

; ---------------------------

; Code for patching the original code in BELL

        org     bellins
        call    chkchar         ; See if key pressed
        jp      nz,bell2        ; If so, cancel BELL command

; ---------------------------

; Patch to backspace-to-rubout conversion function

        org     oldkey

db      rubout

        org     newkey

db      bs

        org     idstr

;       'bs-to-rub conversn'    ; String in standard MEX
db      'rub-to-bs conversn'    ; You must fill same space

        end
```

Listing 2.

```
{
MAP.ZEX

This ZEX5 script runs the XBIOS MAP utility that sets
a drive to a designated virtual disk format.  Since
MAP itself does not allow command-line parameters, ZEX
is used to pass a parameter.  We use foreign formats
only on the 48-tpi drive G: and hence always supply the
input "G".  The argument to the script is the number of
the desired format.
}

b0:map
<G$1
```

To tap into the original code, we choose a convenient point and put in a jump instruction to the patch area. The instruction that we replaced by that jump is the first instruction in the patch area. Then we add the additional code to save the capture buffer free space value into the VALUE variable. Finally, we return to the original code just past the jump we inserted.

The third change implemented in the patch concerns the BELL command. This command is convenient to alert the user to some event. It would be nice to have the bell ring repeatedly until the user signals that he is ready to go on. Unfortunately, the command "BELL 100" will insist on ringing the bell one hundred times. "BELL 0" will insist on ringing it 65,000 times! The only way one can interrupt the bell is to press control-c, but this aborts the entire script. The patch allows any keypress to abort the BELL command without aborting the entire script.

This patch works using a slightly different technique. The BELL code already contains a call to a routine that checks for control-c. We just replace it with a call to a new routine in the patch area that checks for any keypress. After this call, the original code performed a conditional jump to exit from the routine, but for reasons I don't remember, this path was never taken. For our new routine, the conditional has to be switched from a zero check to a nonzero check.

Patching TW.COM

Since we are already on the subject of patching, this seems like a good place for a follow-up discussion of my patch in issue #45 to make TW.COM, the master program of The Word Plus spell checking program, not stop and ask the user if the current configuration is acceptable. When we are running from an alias script, we just want to get on with the job.

Several people tried implementing my patch script and complained that it did not work properly. This was strange, because I have been using it constantly. Hal Vogel finally figured out what was going wrong. It seems that there are two versions of TW in current distribution. Mine, which came with WordStar Release 4, is version 1.21. Other people have a later version, 1.22. Both display a 1981 copyright, and it seems odd that the new version of WordStar would come with an older version of TW. It makes me wonder which version is better.

In any case, version 1.22 has three bytes more code before my second patch point, and so both the address of the poke and the address poked in have to be incremented by three. Here is the original TWPAT alias for version 1.21:

```
get 100 tw:tw.com
poke 103 c3 3b 01
poke 395 c3 2a 04
go $*
```

For version 1.22, only the second poke has to be changed:

```
poke 398 c3 2d 04.
```

In my original discussion, I did not describe how I figured out how to patch around the prompt. Hal Vogel asked about my method, so I will say a little about it here. It's the least I can do to thank him for solving the mystery.

I knew that the offending prompt began with "These are the current settings." So, I searched with a debugger for that text, and I found it starting at address 07FFH. Since it was part of a collection of such messages, all ending with a dollar sign, I knew that they were not part of in-line calls. Therefore, where the message was actually displayed there would almost certainly be an instruction loading the address of the message into one of the double registers.

I used the DSD debugger's search function to locate occurrences of the byte pair FF,07. The first one was part of the code sequence

```
041c    ld   de,07ff
041f    call 17bd
0422    call 17cb
0425    cp   ' '
0427    jp   z,02a4
042a
```

It was pretty clear that 17bd was the routine for displaying a string whose address was in register pair DE and that 17cb was the routine to get an input character from the user. TW takes a space character to indicate that the user wants to make a change, which must be handled by the code at 02a4. Any other character would accept the default setup and continue at address 042a. Therefore, changing the code at 0423 to

```
0423    jp   042a
```

would by-pass the user's response and go right into the spell checking.

However, that change would still send the display of the prompt to the screen. Backing up and putting my replacement jump instruction at address 041d would suppress even the display of the prompt.

As I looked at still earlier code, I found instructions that displayed the current settings. Since we really did not need to have them sent to the screen, I continued to work back until I came to the beginning of that block of code at address 0395, where the message "Summary, Checking file..." was put up. By placing my jump instruction there, all of this extraneous code was by-passed.

You might be wondering how I found the place where the other patch was required. With the main patch installed, TW terminated with an error message about a corrupted file. Guessing that a check of that type would be near the beginning of the code, I just executed TW.COM under the debug-

Listing 3.

```
{
FORMAT.ZEX

This ZEX5 script invokes the XBIOS format program FVCD
and feeds it input to perform the desired diskette
formatting operation.  The following arguments are
recognized:

   <none>       manual operation of FVCD
   F or F:      SB180 DSQD format in drive F
   G or G:      SB180 DSDD format in drive G
   KP2 or KP2:  Kaypro SSDD format in drive G
}

;; SB180 DSQD Formatting

if eq :f: :$1           ;; Test for "F:"
or eq :f :$1            ;; Test for "F"
|if true|
b0:fvcd
<OSXTFF2|cr|
|say|
Formatting in SB180 DSQD format complete.
|end say|
|abort|
|endif|
fi

;; SB180 DSDD Formatting

if eq :g: :$1           ;; Test for "G:"
or eq :g :$1            ;; Test for "G"
|if true|
b0:fvcd
<OSXTGF
|"|
Formatting in SB180 DSDD format complete.
|"|
|abort|
|endif|
fi

;; Kaypro SSDD Formatting

if eq :kp2: :$1         ;; Test for "KP2:"
or eq :kp2 :$1          ;; Test for "KP2"
|if true|
b0:fvcd
<OSXTGF6|cr|
|"|
Formatting in Kaypro 2 (SSDD) format complete.
|"|
|abort|
|endif|
fi

;; Manual Formatting

if eq :$1 :             ;; Test for no argument
|if true|
b0:fvcd
<OSXTGF|until|
|"|
Formatting in interactive mode complete.
|"|
|abort|
|endif|
fi

;; We get here if an illegal argument was given

|"|

-> Illegal format specification given: $1 $2 $3

|"|
```

ger. The problem was clear very quickly.

Some Sample ZEX Scripts

As I mentioned earlier, Bridger Mitchell, author of ZEX version 5.0, covered the specifications for the program in his column in issue #38 of *TCJ*. Having just reread that column, I have to agree with the comments I received that a few examples are needed to appreciate what ZEX can do. I will assume that the reader is already familiar with the basic operation of ZEX and present the only two ZEX scripts that I use regularly.

One of the advantages I got from writing about these scripts is that, naturally, in the course of analyzing them I thought of new approaches and have significantly rewritten them. They do seem to work, but you should be warned that mistakes may have crept in.

If all one wants to do is run a sequence of command lines, I don't think that ZEX is the right approach (though there may be some exceptions that I can't think of now). On the other hand, ZEX is appropriate—in fact, necessary—when one wants not only to invoke a program but also to provide interactive input to the program. ZEX is also useful for creating fancy screen displays, since it has the ability to generate direct output to the console and to suppress output generated by running programs. My two sample scripts fall primarily in the former class.

My first script, MAP.ZEX, is shown in Listing 2. As you see, it is extremely simple. It is used to invoke the MAP utility that is part of the XBIOS extended BIOS for the SB180 computer. This utility can temporarily set each floppy drive to emulate one of a number of foreign formats.

I wanted to have simple alias commands, such as "KP4" or "MD3", that set the appropriate floppy to the Kaypro DSDD or Morrow MD3 format. Unfortunately, MAP is a purely interactive program; choices cannot be passed to it from the command line, and so ARUNZ is powerless. This is where ZEX comes in.

The first section of MAP.ZEX is an extended comment. When a '{' character appears in the first column of a line, all text up to the first closing brace character '}' is treated as a comment and ignored by ZEX. I strongly recommend using comments extensively, as with any other programming language.

The first non-blank, non-comment line invokes the MAP program, which I keep in directory B0:. The next line is the one that does the special job that only ZEX can do. Lines that begin with a '<' character in the first column are interpreted as program input by ZEX, which proceeds to emulate your typing at the keyboard. In this case, the first character emulated is 'G'. The first input that MAP expects is the letter of the drive to configure. Since all the foreign formats I use are for 48-tpi drives, I always use my G drive.

Next, MAP wants one to select a format by number. This we pass to the ZEX script as a parameter on the ZEX command line, and we represent it in the script with the symbol "$1". MAP then wants a carriage return. ZEX is supposed to ignore carriage returns in the script, and so we should include the special directive "|CR|". However, for reasons unknown to me, trial-and-error shows that this is not necessary here; a carriage return is sent automatically whether we want one or not.

MAP.ZEX is an incredibly simple script, but it provides a lot of power. I can now define the following ARUNZ aliases:

```
CLEAR=CLR   zex b0:map 1;msg
KP4=KP10    /xxmap 8 KP4
MD3         /xxmap 9 MD3
TV,803      /xxmap 28 TV803
XXMAP       zex b0:map $1;msg *** Drive G = $2 ***
MSG         echo ^[f$*
```

These scripts implement the nicety of putting a message in the upper right special message area on my Wyse 50 terminal to remind me of the foreign format setting. It is not reasonable to try to remember the special escape sequence for programming this message area, so I provide the alias MSG to do this.

A second subroutine alias, XXMAP, saves space in the ALIAS.CMD file by handling commands common to a number of other aliases. It takes two parameters. The first is the format number for MAP, and it is passed to the MAP ZEX script. The second is the name of the format, and it is passed to the MSG alias. The other aliases, such as MD3, set up the desired disk format. The alias CLR or CLEAR restores drive G to its default SB180 format (MAP selection 1).

The second example script is called FORMAT.ZEX is shown in Listing 3. It works with another XBIOS utility program, FVCD (Format Verify Copy Duplicate). This program can format diskettes not only in the standard SB180 formats but also in several foreign formats. FVCD, like MAP, is a purely interactive program, but I want to be able to format, say, a Kaypro 2 disk by entering just "FORMAT KP2". Again, ZEX saves the day.

This script is much more complicated. Since I want all the formats to be handled by a common alias called FORMAT, the particular format will be designated by a command-line argument and not by the name of the command. This requires testing of the argument, a job that can be performed nicely by a sequence of flow-control commands. That much could be handled by a command alias, but since we will be providing interactive input anyway, we might as well have ZEX take care of everything.

As with MAP.ZEX, we begin the script with a comment block. Note that a double semicolon can be used to enter individual comments on a line. The first two executable lines test the first argument, "$1", to see if it is either "F" or "F:". The colon prefix before the parameter is included to force the argument to be treated as a string and not as a directory specification even if it contains a colon. For reasons I cannot entirely remember, the flow control string-equate testing command does not work properly if it begins with just ":". For that reason, I put the definite string first and the string with the parameter second.

Now we come to a very important ZEX directive: |if true|. This directive tells ZEX to ignore all characters up to the closing |endif| if the current flow state is false. You might wonder why one would need this, since the command processor would ignore commands anyway. The answer is that the script contains material other than commands. It contains interactive input and direct console output. The |if true| and |endif| directives make sure that these are ignored as well.

After ZEX processes a command, such as the line "b0:fvcd", and sends it to the command processor, it looks at the next line. If it begins with a '<' character in the first column, then all the characters on that line up to but not including the ending carriage return are loaded into a key-

stroke buffer and fed, as requested, as interactive input to the running program. In this example, we send "OSXTFF2", just as if we pressed those seven keys manually.

We also send a carriage return, specified by the |cr| directive. As noted with the MAP.ZEX script, ZEX automatically sends one carriage return, but we need two for this operation. We won't bother to explain what all those characters do in FVCD. To figure out what to include, one just runs the program manually and makes note of every key pressed.

This simulated input starts the first diskette formatting. When that operation is complete, FVCD wants more user input. Since the next ZEX line does not begin with a '<', ZEX no longer supplies the input and returns control to the user. More diskettes can be formatted manually.

When one eventually exits from FVCD, the ZEX script continues with the line "|say|". This turns on direct console output. Text, this time including carriage returns, line feeds, and almost all other characters in the file, is sent directly to the screen, until the terminating |end say| directive is encountered. The directive |"|, seen later in the script, is a more compact alternative to |say| and |end say|. It automatically toggles the console output state.

We could have used the ECHO command to display this message, but ZEX's direct screen output is easier and more convenient, as it is case sensitive. You just write things as you want them to look on the screen.

Once we are finished with the formatting, we can terminate the ZEX script immediately. This is done with the |abort| directive. It automatically returns the flow state to its original condition, so we do not have to worry about the fact the we have not executed the "FI" command that balances the "IF" command earlier.

The rest of the script just proceeds to test for other arguments and to pass the appropriate input to FVCD. The only one of these other cases worth commenting on is the manual formatting case near the end of the script. Here we simulate input only to put FVCD in format mode with automatic DateStamper initialization enabled but not to select any format or to start the operation. We must suppress the automatic carriage return that ZEX generates (otherwise formatting will start immediately in some default format). The |until| directive stops simulated input until a carriage return is entered.

My ZMATE Autoexec Macro

The two ZEX scripts were needed primarily because the two programs MAP.COM and FVCD.COM did not provide a facility for designating options on the command line. Ampro always made a particular point of providing such a facility with its utility programs, and I wish all authors would do that. It is nice to have interactive programs, but it should be possible to run them "batch mode" too.

With a word processor or editor one expects the command line tail to include the names of any files to work with. Indeed, this is ZMATE's default action. However, there are many situations in which one would like to do more. For example, one might want to start ZMATE with a macro already loaded into one of the auxiliary buffers. One might even want that macro to be executed immediately. That macro might even carry things to the point of closing the file and exiting from ZMATE! In that way one could use ZMATE to create custom text processors.

How is this done? In an earlier column I described the permanent macro area (PMA) in the ZMATE text editor. This buffer area in the code contains a set of macros that become a permanent part of ZMATE and are available at all times. The first macro in the PMA can be designated as an autoexec macro by placing a control-S in front of it instead of the usual control-X. As soon as ZMATE has loaded and initialized itself—and before it opens any files—it executes this autoexec macro.

Here is a very simple example of a text filter created using this ZMATE facility. Suppose we have a macro that opens a file, goes through it changing each of its characters to upper case, saves the file, and exits. If we put this macro into the PMA as the autoexec macro and then clone ZMATE with the command "XDupcase$", we will have a file called UPCASE.COM. Then we can enter a command like

```
upcase infile outfile
```

to perform the case conversion on a file. This is just a trivial example. With the full facilities of ZMATE macros, one can easily write some very powerful filters.

What I want to describe now is the autoexec macro that I have installed in my standard version of ZMATE (named EDIT.COM). We might call it an indirect macro, since it allows one to specify a macro for automatic execution on the command line. Some sample commands will illustrate how this works.

```
edit source.z80 $ b9e xicomment.mat$ bte
```

The part of the command up to a first dollar sign is taken as the file specification. In this case, the assembler source file SOURCE.Z80 will be opened. In addition, the macro following the dollar sign will be executed. This macro goes to edit buffer 9, reads in the file COMMENT.MAT containing a macro for reformatting assembly code comments, and then returns to the main editing buffer. The following ARUNZ alias can automate this (remember that one needs double dollar signs to represent a single '$' character in the expanded script):

```
ZED,IT  edit $1.z80 $$ b9e xicomment.mat$$ bte
```

Now when I want to work on a Z80 assembler program file called PROG.Z80, I can enter the simple command

```
zed prog
```

Here is a sample command line for editing my ALIAS.CMD file with automatic searching for the alias definition we just showed.

```
aled zed
```

The definition for ALED is

```
ALED if nu $1;edit ram:alias.cmd;else;
edit ram:alias.cmd $$ e s^m$1$$;fi
```

If no search string is given, the alias just edits the file. If a string argument is given, then ZMATE is passed a macro to search for the string at the beginning of a line. The caret followed by an 'm' is interpreted as a control-M or carriage return.

A fully commented version of the autoexec alias is given

in Listing 4. Since I tend to use buffers from both ends, keeping text in the low-number buffers and macros in the high-numbered buffers, for the autoexec macro I use a middle buffer, number 5. The macro switches to it and then inserts the string argument passed on the ZMATE command line.

Listing 4.

```
;Autoexec Macro

; This macro allows ZMATE to be invoked with a macro on the
; command line.  The syntax is as follows:
;
;               ZMATE [infile [outfile]][$<macro>]
;
; The macro line is processed using a special syntax.  All
; letters are normally converted to lower case; characters
; following a '^' are converted to the corresponding control
; character and dollar signs are converted to escape
; characters.  However, a character following a '"' is taken
; literally so that upper case characters, dollar signs, and
; carets can be entered.

B5E                     ; go to buffer 5
I^A:                    ; read in command tail

A                       ; go to beginning
T                       ; tag it
[@T>" _M]               ; move to first non-space, non-
                        ; ...control character
#D                      ; delete the white space
@T=0{                   ; IF nothing left
  BTE                   ;   go to T buffer
  %                     ;   terminate this macro
}                       ; ENDIF

@T="${                  ; IF line starts with dollar sign
  D                     ;   delete it
  0,                    ;   push 0 (false ) onto stack
}{                      ; ELSE
  -1,                   ;   push -1 (true) onto the stack
  E                     ;   suppress error trapping
  S$$                   ;   search for a '$' separator
  @E{                   ;   IF none found
    Z                   ;     go to end of buffer
  }{                    ;   ELSE
    -D                  ;     delete the '$' separator
  }                     ;   ENDIF
  #B0M                  ;   move file name specs to buffer 0
}                       ; ENDIF

[                       ; REPEAT
  @T<"A!(@T>"Z)'{       ;   IF upper case letter
    @T!96R              ;     replace with lower case
    ^                   ;     loop back
  }                     ;   ENDIF
  @T=""{                ;   IF character is double quote
    D                   ;     delete it
    M                   ;     move past next character
    ^                   ;     loop back
  }                     ;   ENDIF
  @T="^{                ;   IF character is caret
    D                   ;     delete it
    @T&31R              ;     convert next char to control
    ^                   ;     loop back
  }                     ;   ENDIF
  @T="${                ;   IF character is dollar sign
    "$R                 ;     replace it by escape char
    ^                   ;     loop back
  }                     ;   ENDIF
  M                     ;   (otherwise) move past the char
@T=0]                   ; UNTIL end of text

BTE                     ; go to T buffer
@S{                     ; IF stack was true
  XF^A@0$               ;   open the file spec in buffer 0
}                       ; ENDIF

B0K                     ; clear buffer 0
.5                      ; execute macro in buffer 5
B5K                     ; clear buffer 5
```

ZMATE has the special string symbol control-A-colon for this. [In the listing, control characters are indicated by leading carets, and escape characters are shown as dollar signs. However, some of the characters really are dollar signs, and you will have to determine which are which by the context.]

The next block of code in the macro removes any leading space. If there is then nothing left, the main text buffer is reselected, and the macro is aborted.

The next task is to split any arguments passed into those that specify files and those that comprise a macro to execute. The code begins by checking whether the first character is a dollar sign and saving this information on the stack. If the first character is '$', then no edit file was specified, and the rest of the line is a macro. Otherwise, we try searching for a dollar sign. If one is found, everything before it—which specifies the files to open—is moved to buffer 0. If none is found, we will be at the end of the buffer, and the entire text will be treated as the file specification.

The next block of code performs some special interpretation on any macro command that remains. Since the CP/M command line imposes some limitations on the characters that can be passed, we provide means for indicating those that cannot be put into the command line. When a double quote character is found, it is deleted, and the next character is allowed to pass without interpretation. Otherwise, the following special conversions take place: (1) all alphabetic characters are converted from upper case to lower case; (2) dollar signs are converted to escape characters; (3) a caret is deleted and the character after it is converted into the corresponding control character.

Once the macro in buffer 5 has been fully interpreted, we switch back to the main editing buffer. If the Boolean value on the stack is true, then we open the files (using the "XF" macro) specified by the string in buffer 0. I don't think we talked about this before—at least not in detail—but this is an example of the way ZMATE can use the contents of buffers as string arguments for other macro commands. This greatly enhances ZMATE's power.

The macro completes its work with three more steps. Buffer 0 is cleared out. Then the macro in buffer 5 is executed. Finally, buffer 5 is cleared out. That's all for the macro, and that's all for this column! See you again next issue.•

Wanted:
DEC TU-80 Tape Drive
with Unibuss Controller for
VAX 730

Am interested in either purchase or trade

Contact:

Billy D'Augustine
95 East Central Avenue
Wharton, NJ 07885

or usenet: billy@westmark.com

Continued from Page 3

Graphics can be submitted in either TIFF or EPS formats. In time we will be getting a more complete set of tools for handling advanced graphics. One piece I see a particular need for is Schema. Until that day arrives, play it safe with graphics. Always send a hard copy printout.

We can handle MS-DOS disks in 5.25" 360k, and 3.5" 720k and 1.44M formats, and any soft sectored 5.25" CP/M soft sectored format, including quad density. If you are uploading, please ZIP the files or make a library of them. Socrates Z-Node is at (908) 754-9067, at up to 2400 bps. Starlink users can use the 3319 outdial.

Continued from Page 31

label files such as -UTILITY.001, however, Gene has made SAP-Z configurable to erase all, keep all, or to keep only those disk-label files which begin with a certain tag character. A .CFG configuration file is supplied for use with ZCNFG.COM.

Another smart feature is SAP-Z's preservation of date stamps. The DateStamper !!!TIME&.DAT file is also sorted and rewritten to keep all date stamps in proper order.

To sort and pack drive D:, then, I ran SAP-Z thus...

```
A0:BASE>SAPZ D:
 Sorting and packing drive D:
  -> Reading, Sorting, Writing, Saving Dates, Relogging, Done
```

The D: directory now lists files alphabetically, speeding up access by DIR, SD and other DIR programs. And, since we've "cleaned" the directory of erased files, the problems, which sometimes arise in using UNERASE on directories containing a large number of erased files, have also been minimized.

LBREXT31 (ZSUS Vol 1 #12)

Howard Goldstein (Z program debugger extraordinaire) has made a number of improvements to the LiBRary EXTraction utility over the past year (between versions 2.6 and 3.1). LBREXT is great for quick extraction of files from LBRs when you don't want to invoke the larger LBR managers (VLU, NULU). Many users automate transfer of files between LBRs using LBREXT and LPUT under control of an alias.

As with many utilities updated this year, LBREXT has been linked with Version 4 of the Libraries, including the DSLIB time and date stamping routines, and now can handle date stamps from ZSDOS, DateStamper, a LBR's DRI-style member date, or CRUNCH (vs 2.3d)-style imbedded datespecs. Another improvement has been the addition of CRC checking (older versions would not extract files from a LBR if there were no CRC's in the directory). Now, if the CRC doesn't match that of the file, instead of aborting, a warning message is issued and the file is extracted nonetheless. Finally, (and another trend in recent updates) is the addition to the LBREXTxx.LBR of a .CFG configuration file for use with ZCNFG, Al Hawley's fine Z-System configuration program. Now you can avoid the trepidation of manually patching LBREXT to install default parameters.•

And Now, Introducing...

We introduce several new authors in this issue. Carson Wilson gives a review of "BDS Z". Carson is one of the authors of ZSDOS, the modern replacement to the DRI BDOS for CP/M. Certainly not the sort of fellow to shy from assembly programming. If he has taken an interest in this version of C, it must be pretty good! By the way, Carson tells me his ZSDOS Programmer's Guide is being very well received. Don't delay in ordering your copy.

Wayne Sung shows how to interface an Iomega Bernoulli drive to a CP/M computer. In so doing, he builds a SCSI interface! That should keep you busy a while. This leads to a much wider topic: how does one go about installing generalized I/O with SCSI on existing equipment without a SCSI port? Wayne's approach certainly deserves your scrutiny.

Bill Tishey joins us with a run down on the latest Z-System software. Bill is the *Z-System Software Update Service* editor and for good reason. You should see his Z-System software database! He lists hundreds upon hundreds of programs. Bill is undoubtedly the pre-eminent authority on Z-System software availability and usage.

Clif Kinne joins us to explain the ZMATE macro language. Think of ZMATE as a programming language for text and you can imagine how far this can lead. We will be seeing more of Clif.

Enough rambling. There is plenty in this issue to occupy your winter evenings. Let's get on with it. Enjoy yourself. This, bud, is for you!•

Cross-Assemblers as low as $50.00
Simulators as low as $100.00
Cross-Disassemblers as low as $100.00
Developer Packages
as low as $200.00(a $50.00 Savings)

A New Project
Our line of macro Cross-assemblers are easy to use and full featured, including conditional assembly and unlimited include files.

Get It To Market--FAST
Don't wait until the hardware is finished to debug your software. Our Simulators can test your program logic before the hardware is built.

No Source!
A minor glitch has shown up in the firmware, and you can't find the original source program. Our line of disassemblers can help you re-create the original assembly language source.

Set To Go
Buy our developer package and the next time your boss says "Get to work.", you'll be ready for anything.

Quality Solutions
PseudoCorp has been providing quality solutions for microprocessor problems since 1985.

BROAD RANGE OF SUPPORT
- Currently we support the following microprocessor families (with more in development):

Intel 8048	RCA 1802,05	Intel 8051	Intel 8096
Motorola 6800	Motorola 6801	Motorola 68HC11	Motorola 6805
Hitachi 6301	Motorola 6809	MOS Tech 6502	WDC 65C02
Rockwell 65C02	Intel 8080,85	Zilog Z80	NSC 800
Hitachi HD64180	Motorola 68000,8	Motorola 68010	Intel 80C196

- All products require an IBM PC or compatible.

So What Are You Waiting For? Call us:
PseudoCorp
Professional Development Products Group
716 Thimble Shoals Blvd, Suite E
Newport News, VA 23606
(804) 873-1947 FAX: (804)873-2154

Adding a Bernoulli Drive to a CP/M Computer

By Wayne Sung

No matter how large a disk drive you have, eventually it will be full. No matter how much you paid for a drive, seemingly somewhere along the line it crashes. To get decent access times you have to buy a much larger drive than you really wanted. Then there's the question of backup...

The Drive

Iomega has been making a family of drives called Bernoulli drives for close to ten years now. These drives answer all the above concerns. They are removable cartridge drives, so if one cartridge gets full you put in another one (and you get to control-C it!) [Ed. unless, of course, you are using a modern replacement DOS like ZSDOS, which handles this automatically].

The Bernoulli technology means that the recording surface is allowed to deform. As a result, particulates do not cause head crashes, just soft errors. With the advanced error correction in the drives, you might not even notice the soft error.

The Alpha drive, Iomega's first product, is a 10 Megabyte drive with 35 millisecond average access time. Although this is not a current product (they now make drives of up to 40 Megabyte capacity) the 10 Megabyte size is a good fit for CP/M 2.2 systems. Since the disks are removable, backup is just like any other floppy backup, especially if you have two drives. These drives are now beginning to show up in the surplus market, and I finally have a decent mass-storage system.

Physically, the drive is the same size as a full-size 8-inch floppy drive. It even looks like one. The power requirements are a little different, so you can't generally use the same disk boxes. I bought a dual drive cabinet ready to run except for an interface cable. The box came with a DC37 connector, which I removed to run a 50-conductor ribbon cable directly to the connector on the drive.

On the drive are a number of setup switches. The one that will probably have to be changed is the parity switch (I don't use parity; the drive came expecting it). There are also switches for unit number, termination, and SCSI select number. There is normally one master unit and up to three slave units. The master unit has all the interface electronics, both analog and digital. The slave units only have a motor control board. There are actually two sets of terminators, one for SCSI and one for the internal chain.

The Port

The hardware of a SCSI port has three sections. There are 8 bi-directional data lines with an optional parity line. Then there are two sets of unidirectional control lines. SCSI has the capability of multi-master operation, that is to say, each SCSI controller can direct any other on the same port. For our purposes I will bypass that capability. It simplifies the hardware if you want to make your own port, and it simplifies the software considerably.

There is also a disconnect capability, where, for example, if you have a combined tape and disk drive you could command a disk to tape backup and let it go. It will notify you when it's finished. This feature is not often available in controllers, so it also will be sidestepped.

I had built a number of other interfaces using an 8255 parallel port, and this port is also done with one. Some machines already come with SCSI ports, and on those all you would have to do is plug the two together.

The lines that go from the computer to the drive are RST (reset), SEL (select), and ACK (acknowledge). The lines that go from the drive to the computer are C/D (command/data), I/O (input/output), MESG (message), REQ (request), and BSY (busy). There is one more line called ATN (attention) from the drive to the computer to signal exceptional conditions (including the "return from disconnect" mentioned above). This signal is not used on the Alpha drive. All signals are active low. The sequence of operations is as follows.

Upon power up you would toggle the reset line to the drive. This is not absolutely necessary but does not hurt. To begin accessing the drive, first you put the value of the select code on the data lines. There can be up to 8 devices on the port, corresponding to the 8 data lines. Only one line is used to identify the device. There have been proposals to allow devices to decode the entire 8 bit word for a device address, but generally devices only use the single bit. A controller can address any number of sub-devices, though.

Next you assert the select line to the drive and wait for it to assert the busy line. At this point you are ready to talk to the drive. If this is the first time the drive has been used after power up, it is good to send it a rehome command (the command formats will be described later). This command also clears out any error messages which may have accumulated, including a somewhat unknown condition after power on.

To send the command block, the C/D line will be set to command mode, and the I/O line will be set to input mode. Then the REQ and ACK lines handshake each byte across. Note that

Wayne Sung has been working with microprocessor hardware and software for over ten years. His job involves pushing the limits of networking hardware in attempting to gain as much performance as possible. In the last three years he has developed the Gag-a-matic series of testers, which are meant to see if manufacturers meet their specs.

whether the data bus is writing to the drive or reading from the drive, it is always the drive that asserts REQ and the computer that returns ACK. A command block is usually six bytes. At the end of the command block C/D changes to data mode, and I/O will change or stay the same depending on the direction of data.

If there is no further data, one byte of status becomes available. This indicates whether the previous operation was successful. Then MESG asserts, indicating the last byte of the exchange. At this point all control lines go back to their rest state. Note that the controller must be reselected for each exchange.

In my port, ACK is generated in hardware. Each time I use the data port I send an ACK to the drive. REQ is polled and is connected to the high bit, so I can use a rotate to get the bit value. The other lines can be arranged in any order, since each time you read them you would be comparing against a mask anyway.

I had a controller once that was fast enough that I could use the INIR and OTIR instructions to read or write the data blocks. Unfortunately, most controllers are not that fast, and you do have to synchronize the transfers. If your hardware includes a SCSI chip, then of course all this handshaking is done for you, and you won't have to program it yourself.

Referring to the schematic diagram in Fig. 1, we see that the 8255 has three ports. I use the first port for the data bus so that the two address lines can both be 0 during data bytes. This makes the hardware ACK easier. The second port is the input to the computer from the drive, and the third port is the opposite.

A fourth port exists for configuring the 8255. This port is also used during operation, as the data port has to change direction often. The buffers for the port are not exactly SCSI spec, but the port works well enough for two devices.

The control line buffers are all one-way, so they are simply inverters. The data buffers have to be two-way, and the direction change is actually controlled by the I/O line from the drive. Note that I used 74LS243's for the data lines simply because that's what I had on hand. You can use a 74LS244 or 74LS245 just as easily. If you use an inverter, then you eliminate the need for inverting the data in software. The control line drivers should be inverters because the 8255 resets with all lines low. If there is no inversion, then the SCSI side will look as if all lines are asserted.

The automatic ACK circuit works as follows. The data port on the 8255 is at the lowest of a four-address group. Thus both A0 and A1 to the chip will be low when the data port is accessed. One of the four sections of a 74LS02 checks for this condition. Of course, either read or write accesses count, so three sections are used to 'OR' those two together. However, the select process also uses the data port but should not generate an ACK. So one section of a 74LS74 (the other section is not used) is set by a REQ pulse from the drive. Combining all these conditions gives the ACK pulses only for data port accesses that are in response to REQ pulses.

The Device Driver

Certain characteristics of the drive determined the best way the device driver should be written. Note that with SCSI devices, accesses are by 'logical block' and not by track and sector. In fact, with this drive you really have no way of knowing what track and sector you are accessing, because there can be bad block replacements going on that are com-

Listing 1. Driver code for the Bernoulli drive.

```
; Addresses of the parallel ports

data    equ     0fch
stat    equ     0fdh
ctrl    equ     0feh
setup   equ     0ffh

; Commands for setting the direction of the data port

setout  equ     082h
setinp  equ     092h

; Output command bit values

select  equ     1
reset   equ     2

; Input bit mask values

req     equ     1               ;probably use rar instead
cd      equ     2               ;bit high = command
                                ;..low = data
io      equ     4               ;high = in from controller
mesg    equ     8               ;completion flag
bsy     equ     80h             ;probably use ral instead

; These are variables that the driver needs. The values are
; for my setup

ERRFLG  EQU     0DE18H
HSTBUF  EQU     0E700H

; Values used in constructing the DPB. Deblocking code also
; has to know them. The physical sector size requires a hack
; in calculating the SCSI block number.

HOSTSZ  EQU     512
CPMSPT  EQU     256
BLKSIZ  EQU     16384

; The command values are complemented because of the
; hardware actual data on disk will also be complemented but
; this is invisible.

rddisk  equ     0f7h
wrdisk  equ     0f5h
ctladr  equ     0feh            ;controller address is 1


; These are the rdhost and wrhost calls that can be added to
; a bios. Input values needed are track, sector and where
; the data is or is going.

rdhost: call    cnvsec          ;change track/sector to
                                ;..logical block
        call    selctl          ;select the SCSI controller
        jnc     done            ;if select failed return
                                ;..error
        mvi     a,rddisk        ;put read command into
                                ;..command block
        sta     comtbl
        call    send6           ;send command block to
                                ;..controller
rdla:   in      stat            ;check for the controller
                                ;..response
        cpi     bsy+io+req
        jnz     rdla            ;wait for the proper state
        mvi     a,setinp
        out     setup           ;turn data port around
        lxi     h,hstbuf
        lxi     b,0200h         ;512 bytes per sector
rdloop: in      stat
        rar
        jnc     rdloop
        in      data
        xri     0ffh
        mov     m,a
        inx     h
```

```
        dcr     c
        jnz     rdloop
        dcr     b
        jnz     rdloop          ;move the data in
        jmp     chkerr          ;see if xfer was error free

; The comments for the wrhost call are essentially the same
; as for rdhost.

wrhost: call    cnvsec
        call    selctl
        jnc     done
        mvi     a,wrdisk
        sta     comtbl
        call    send6
wrla:   in      stat
        cpi     bsy+req
        jnz     wrla
        lxi     h,hstbuf
        lxi     b,0200h
wrloop: in      stat
        rar
        jnc     wrloop
        mov     a,m
        xri     0ffh
        out     data
        inx     h
        dcr     c
        jnz     wrloop
        dcr     b
        jnz     wrloop          ;move the data out

; Common check for status and message bytes. B is set non-
; zero if there is an error or 0 if no errors. This will
; then be stored in ERRFLG.

chkerr: mvi     c,bsy+cd+io+req
        call    rechk
        jnc     chk1            ;reached the status state
        mvi     b,2             ;any value other than 0
chk1:   call    check1
        mov     b,a             ;this is actual status byte
        mvi     c,bsy+mesg+cd+io+req
        call    rechk
        jnc     chk2            ;reached the mesg state
        mvi     b,4             ;any value other than 0
chk2:   call    check2          ;message is always 0

; The return value includes the drive number in the chain.
; Error, if any, is in the low bits so the drive number can
; be removed.

done:   mov     a,b
        ani     7
        sta     errflg
        ret

; Logical block number = track number * 64 sectors/track +
; sector number except two logical blocks are available in
; each sector. CP/M sector number has been changed to host
; sector number at RWOPER, but we want to double the logical
; block number presented to the drive because both blocks
; are read each time. This is not two separate reads, rather
; it is a single read calling for two blocks of data.

cnvsec: lhld    hsttrk          ;get CP/M track value and
                                ;..multiply by 64
        dad h ! dad h ! dad h   ;six double adds
        dad h ! dad h ! dad h
        lda     hstsec
        mov     e,a
        mvi     d,0
        dad     d               ;add the CP/M sector number
        DAD     H               ;one more double - final
                                ;..result is
                                ;..track*128 + sector*2
        mov     a,l
        xri     0ffh
        sta     comtbl+3        ;put the values in the
                                ;..command block
        mov     a,h
```

pletely invisible to you. Thus the CP/M disk declarations are more for the purpose of having your STAT DSK: display give you something close to what you would expect, rather than corresponding one-for-one with the physical characteristics.

For example, you could have the drive declared as 65536 sectors on one track. Since everything has to be converted to logical block numbers to the drive, it won't matter. My final declaration looks like 256 tracks of 256 sectors (CP/M sectors, not physical). The physical mapping of the sectors presented a problem. The disk sectors are 512 bytes but are presented as two logical blocks.

The easy way would have been to use 256 byte physical sectors, but this resulted in rather slow transfer rates. I was able to get a much better transfer rate by reading both blocks at once, making CP/M think the disk has 512 byte sectors. However, and you will notice this in the driver code, I had to rig the block numbers to make everything come out right (for each CP/M disk read it thinks it's reading one 512 byte record but actually it's reading two 256 byte records).

Except for this bit of trickery, the device driver is pretty straightforward and runs from the read and write host call points of the CP/M deblocking code. The driver I use is shown in Listing 1.

A lot of the code is simply for synchronizing the control lines. If you have a real SCSI port, the code should be a lot shorter. Note that there is no error retry in this code. The drive would have done sufficient retries on its own. If it comes back reporting errors, that will be pretty much final. Also no effort was made to extract the exact error condition if one is returned. Not enough errors occur to make the effort worthwhile.

I would like to digress for a moment to talk about the construction of the DPB and how different items affect each other. I have installed many DPBs, but usually they are already done and I don't think about why a given DPB may have been done that way. Going through this one in a little more detail allowed me to realize that there is considerable interplay among the entries.

To begin with, there is a 16 bit record pointer, which means the maximum number of CP/M sectors on a disk is 65536 128 byte sectors or 8 Megabytes (CP/M 3 and other systems have extended the record pointer so as to allow larger disks). There is actually no notion of tracks in the DPB. The sectors-per-track entry and the total number of host sectors on the drive are used to derive which track a given sector should be on.

How would one chose the block size, i.e., how many CP/M sectors per block? Smaller block sizes are more efficient. The minimum allocation is one block, so that much is used whether a file is one byte or exactly one block. Small blocks, however, require a large allocation vector. One bit in the allocation vector represents one block on the disk, so an 8 Megabytes drive using 2k blocks would require 4096 bits or 512 bytes of allocation vector. This is a lot of memory to give up, especially if you don't have a banked system. Also, smaller blocks mean that large files would require more directory entries, which in turn means that the total number of directory entries allowed should be increased. This winds up increasing the size of the directory check vector. Of course, you could omit directory checking altogether (which I did), but it is removable...

Not evident from the code is the fact that the physical disk has been reformatted with a 32 sector skew (on a 64 sector

track). This allows access to two sectors per revolution and is about the highest rate the computer can handle. Many systems that claim no interleave are merely ensuring they get to at least one sector per revolution. It is quite difficult to put away a sector in the time between sectors. If you had enough memory, you could read a whole track at once and then

distribute it later. Not having access to the actual track and sector numbers, though, means you really can't do this effectively.

Note that the drive maker tried to help access time by deliberately offsetting the beginning of successive tracks, since when you read past the end of one track the step necessary would have caused a number of sectors to go by. This offset is not removable and I really can't say whether it makes a whole lot of difference in a simple operating system. Also, the second byte of the command block can include another few bits of the block number but we don't have that many.

If you are interested in pursuing this project yourself, you may get copies of this listing and also of a formatter program by sending a labeled, formatted diskette with a return mailer and postage plus $1 handling to Jay Sage (see the SME ad for the address).•

Listing 1. Continued

```
        xri     0ffh
        sta     comtbl+2
        ret

send6:  mvi     b,06h           ;the command block is always
                                ;..6 bytes
        lxi     h,comtbl
send6a: in      stat
        rar
        jnc     send6a
        mov     a,m
        out     data
        inx     h
        dcr     b
        jnz     send6a          ;send the command block
        ret

; This call compares the input leads with an expected value.

rechk:  lxi     d,0             ;timeout was included so the
                                ;..call won't lock the
rechk1: in      stat            ;..machine if something went
        cmp     c               ;..wrong. It has not proven
                                ;..to be useful.
        rz
        dcx     d
        mov     a,e
        ora     d
        jnz     rechk1
        stc                     ;set carry if error
        ret

check1: mvi     a,setinp        ;change port direction
        out     setup
check2: in      data            ;get error byte
        xri     0ffh            ;see if 0
        ret

selctl: mvi     a,setout
        out     setup           ;be sure data port direction
                                ;..is correct
        mvi     a,ctladr        ;controller is reselected
                                ;..each command
        out     data
        mvi     a,select
        out     ctrl
        xra     a
        out     ctrl            ;select controller
        in      stat
        ral
        mvi     b,1             ;non-zero for select error
        ret                     ;carry set if selected

; This is the command block. the values for unit and blocks
; do not change. This data is complemented due to the
; inverted data bus.

comtbl: db      0ffh            ;opcode goes here
        db      0dfh            ;unit number goes here (I
                                ;..use 2nd unit)
        db      0ffh            ;high byte of the logical
                                ;..block number
        db      0ffh            ;and low byte
        db      0fdh            ;two blocks every time
        db      0ffh            ;extension byte - always 0

; These can be filled by the SETTRK and SETSEC calls or if
; you know where these actually are those can be used
; instead.

HSTTRK: DW      0
HSTSEC: DB      0

; 64 512-byte sectors per track, 306 tracks, 256 directory
; entries, NO CHECK, no system tracks

DPB:    DW      256             ;64*(512/128) records/trk
        DB      07              ;128*(2**7) = 16K blocks
        DB      7FH             ;block mask (01111111)
        DB      07              ;16K per directory entry
        DW      512 - 1         ;highest block number
        DW      256 - 1         ;highest directory entry
        DW      80H             ;initial allocation vector
        DW      0               ;bytes in directory check
                                ;..use 64 for full checking
        DW      0               ;reserved track offset
        DB      0
        end
```

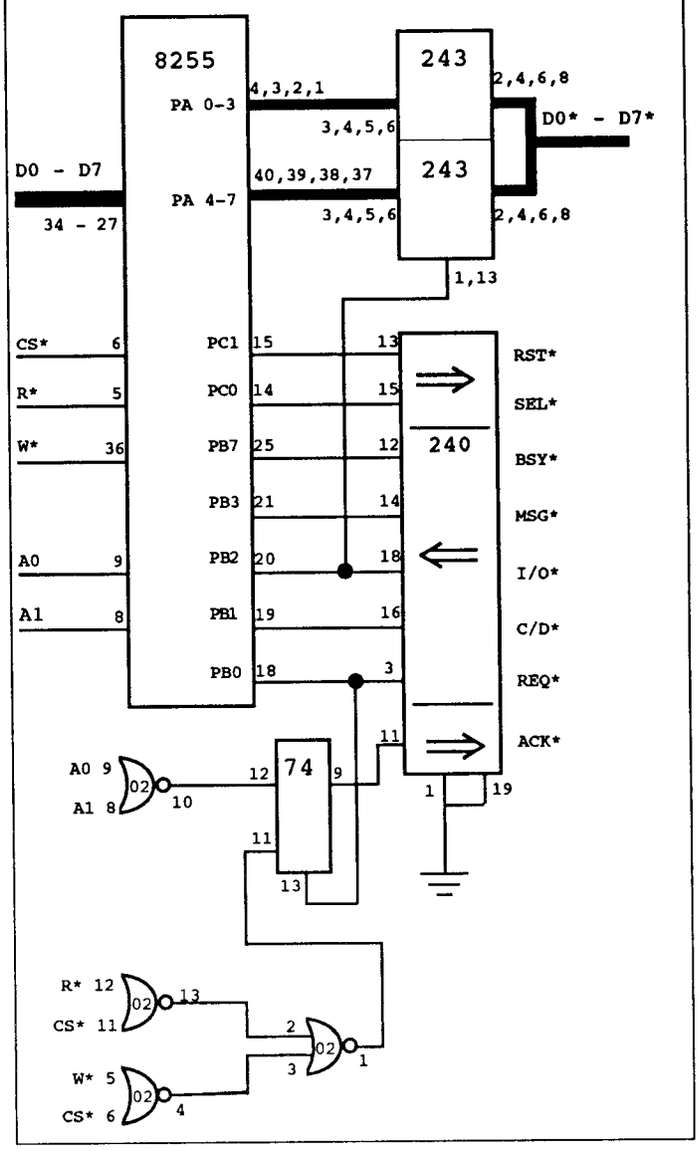
Listing 2. Formatter for Bernouilli drive

```
; This is the formatter for the IOMEGA ALPHA drive, giving a
; 32 sector interleave. Cartridges are formatted from the
; factory so this step can be omitted. The operation occurs
; offline and completes in less than 5 minutes. Note that if
; you replace the format command with another command this
; becomes a standalone command sender. For example, the
; recal command may be needed if your setup seems to power
; up in a somewhat unknown state.

; The command values are complemented because of the
; hardware.

recal   equ     0feh    ;rehome the drive
format  equ     0fbh    ;format entire drive

ctladr  equ     0feh

; The code here is essentially the same as the main device
; driver.

data    equ     0fch
stat    equ     0fdh
ctrl    equ     0feh
setup   equ     0ffh

setout  equ     082h
setinp  equ     092h
select  equ     1
```

```
reset   equ     2

req     equ     1               ;probably use rar instead
cd      equ     2               ;bit high=command, low=data
io      equ     4               ;high = in from controller
mesg    equ     8               ;completion flag
bsy     equ     80h             ;probably use ral instead

        org     100h
        jmp     start

send6:  lxi     b,0600h+data    ;b = count, c = port
        lxi     h,comtbl
send6a: in      stat
        rar
        jnc     send6a
        mov     a,m
        out     data
        inx     h
        dcr     b
        jnz     send6a
        ret
rechk:  lxi     d,0
rechk1: in      stat
        cmp     c
        rz
        dcx     d
        mov     a,e
        ora     d
        jnz     rechk1
        stc
        ret
check1: mvi     a,setinp        ;change port direction
        out     setup
check2: in      data            ;get error byte
        xri     0ffh            ;see if 0
        ret

selctl: mvi     a,setout
        out     setup           ;be sure data port direction
                                ;..is correct
        mvi     a,ctladr        ;controller is reselected
                                ;..per command
        out     data
        mvi     a,select
        out     ctrl
        xra     a
        out     ctrl            ;select controller
        in      stat
        ral
        ret                     ;carry set if selected

start:  call    selctl
        jnc     error
        mvi     a,0DFh
        sta     intrlv          ;interleave=32
        mvi     a,format
        sta     comtbl
        call    send6
        mvi     c,bsy+cd+io+req ;these lines high when
                                ;..command done
ckfmtd: call    rechk
        jc      ckfmtd
        call    check1
        mvi     c,bsy+mesg+cd+io+req
        call    rechk
        jc      error
        call    check2
        jnz     error
        ret
error:  hlt             ;whatever you might want here.
                        ;..Format errors hardly ever happen
                        ;..on new disks.

; This data is complemented due to the inverted data bus.

comtbl: db      0ffh            ;command byte
        db      0dfh            ;unit number
        db      0ffh            ;not used for format
        db      0ffh            ;not used for format
intrlv: db      0ffh            ;interleave code - otherwise
                                ;..normally 0ffh
        db      0ffh            ;not used
        end

; This is a fragment showing how to reset the port. If you
; are using an 8255 this should be run before the drive is
; used. Port addresses are the same as above.

start:  mvi     a,setout
        out     setup           ;set 8255 pa=in, pb=in,
                                ;..pc=out
        mvi     a,reset
        out     ctrl
        xra     a
        out     ctrl            ;reset controller
        lxi     b,1000h         ;give it some time to do it
delay:  dcx     b
        mov     a,b
        ora     c
        jnz     delay

        end
```

Figure 1. **SCSI**

All unused inputs are pulled high



Z Best Software

A Look at New Z-System Programs

By Bill Tishey

Hi everyone! Welcome to this new section on Z-System software. Readers may remember *TCJ*'s ZSIG and ZCPR3 columns several years ago in which Jay Sage outlined new ZSIG program releases. This column might be considered a revival of that concept. Each issue I will be bringing you some brief views of Z-System programs and utilities available in the public domain (and distributed through the Z-System Software Update Service). I'll describe new releases, as well as the on-going improvements being made to existing utilities, and from time to time will try to relate trends in Z-System program development in general.

Who Am I, You Ask?

First, I should make it clear that I am not a programmer myself, just a user who has "grown up" with Z (from ZCPR2 to the current Z34/NZCOM), and done some simple computing along the way (file-patching, assembly of system segments, alias-building, etc.) to get my system to work *for* me vice *against* me. Over the years, however, I've gained some familiarity with much of the software available for ZCPR and Z-System. For the past few years I've been what you might call the "Z-System Librarian", comparable to Jay's role as original "Z-SIG Librarian." Jay reviewed new, user- generated software and organized it into official ZSIG libraries. I've also been collecting and reviewing new program releases and organizing them for distribution through the new Z-System Software Update Service. As a result, I've developed a rather comprehensive library and database (over 700 items at last count) of existing Z software.

A firm believer in program documentation, I've also built a rather extensive system of help files to go along with these many programs and utilities. The process of creating .HLP files has been very educational and I hope I can pass on to Z users what I have learned. My objective will be to familiarize you with the best and most useful of existing programs. At the same time, I will be taking requests. If you've heard of a Z program which you would like discussed, please let me know. I'd be happy to fit it in.

Z Happy Programmers...

I thought it appropriate in this first column to highlight some of the programmers who bring you "Z Best Software". These aren't in any special order, although I've tried to place Z-System "developers" first. This list certainly isn't exhaustive. There are many Z enthusiasts who share their creations with others. These are simply some of today's most active programmers with whom I'm familiar (I apologize if I've overlooked anyone). Many of the programs listed (and others) will be discussed in future columns. Note that the numbers in each filename indicate the current version of the program. In most cases the programmer is the original author of the program, but in some instances may be responsible only for the latest improvement or update (a subject for a future column!).

Jay Sage. Jay is the author of ZCPR34 and architect of the NZCOM and Z3PLUS autoinstall Z-Systems. Besides his highly popular alias expander (ARUNZ09U) and file shell (ZFILER10O), Jay has produced many system-related utilities throughout the development of ZCPR. [CLRRSX11, COMIF10, ERRSET13, NZEX-D, PAUSE11, SHOW14, VALIAS2B, XECHO10, XTCAP10, Z33ERR08, Z33IF14, ZBGQK11, ZMSAVE11]

Bridger Mitchell. Bridger is well-known as the author of DateStamper, BackGrounder ii, JetFind, and DOSDISK. His Z3PLUS (Z-System for CP/M Plus computers; see *TCJ* #33) was a major contribution which allowed porting of Z to many other Z80 machines. [FIXT&D, JETLDR10, JLTOOLS, V06, ZEX5]

Joe Wright. Joe is the author of NZCOM (and its predecessor Z-COM) and holds the source code for the Z-System Libraries. His efforts have helped to ensure standardization and compatibility among programs in the Z-System environment. Joe has produced the B/PRINTER, I/O RECORDER and NUKEY input/output packages, the NZ-TOOL4 and NZ-TURBO Turbo Pascal interface packages, and many other system and program enhancements. [ALIAS15, DIR14, MENU41, NZBIO15, NZTIM, NZ-WS4, PATH31, SAVNDR13, ZEX404]

Harold Bower. Hal is a member of the ZDOS development team and,

Bill Tishey has been a ZCPR user since 1985, when he found the right combination of ZCPR2 and Microsoft's Softcard CP/M for his three-year-old Apple II+. After graduating to ZCPR30 and PCPI's Applicard CP/M, he did a "manual install" of ZCPR3.3 (with help from a lot of friends!), and in late 1988 switched to NZCOM and ZSDOS, all on the same vintage Apple II+. Bill is the author of the Z3HELP system, a monthly-updated system of help files for Z-System programs, as well as comprehensive listings of available Z-System software. Bill is the editor of the Z-System Software Update Service and has compiled such offerings as the Z3COM package and the Z-System Programmer's Toolkit. Bill is a language analyst for the federal government and frequents the Foreign Language Forum (FLEFO) on Compuserve. He can be reached there (76320,22), on Genie (WATISHE), on Jay Sage's Z-Node #3 (617-965-7259) and by regular mail at 8335 Dubbs Drive, Severn, MD 21144.

along with Joe Wright, has been entrusted the source code for the Z-System Libraries. Hal continues to improve on SYSLIB, Z3LIB, VLIB and DSLIB (vs 4.3 should be available by the time you read this). He is also working on a new-generation MCAT/XCAT (a generic cataloger for CP/M 2.2, CP/M Plus and Z-System). [MYLOAD, PDMSHELL, SPEEDUP12]

Cameron Cotrill. Cam is the architect of ZSDOS/ZDDOS and a great organizer and conceptualist. He continues to provide many programming and system-level enhancements to Z-System. [CFORZ02, NZBLTZ10, Z33TRC11, TXT2DB10, and TRAP02]

Carson Wilson. Carson is a member of the ZDOS development team, author of the acclaimed ZDE Display Editor, and is responsible for much of the program development in support of Z3PLUS. [BU17, CALRCP11, CD39, FORZ10, CPA12, DIFF30, DIRATR11, DU314, EXTEND13, NZFCP13, JETCP10, LUSH10, PARMLIB1, PPIP19, TAILZ10, ZLT13A, ZSLIB21, ZTYPE10]

Al Hawley. Al is the author of the ZAS and ZMAC assemblers, REVAS debugger, and the Z-System configuration program ZCNFG. Over the years, he has provided many tools and continuing advice and support for Z-System programmers. [ARRAYS, TCLOCK11, EDITND11, LOADND12, PWD20, RLIB12]

Howard Goldstein. Howard has the reputation of being the premier Z-System software sleuth, bug finder and fixer. In addition to his many hours of debugging and keeping programs efficient and working to standard, Howard still finds time for support and advice to others (both budding and experienced programmers alike). [Z33IF15, MOVE22, REG13, RESOLV14, SAVE15, SHFILE11, SHVAR13, XD13C, Z3LOC18, Z3VARS2, ZCRCK13, CPSET13, FF20, LBREXT31, LDIR-B16, LBRHLP16, LX21, TLINE12]

Rob Friefeld. Rob is well known for his screen-oriented alias builder (SALIAS15) and his outstanding command line editing shell (LSH10R). He continues to improve on his other screen-oriented utilities and to find new and innovative applications for Z-System (note his bevy of RCP alternatives). [BCOMP21, SREN10B, VCOMP21, VIEW43, RCPZRL11, DEBUG11, RCPCALC, RCPMC, RCPPEEP DIRBAR, NT46, SNAP12, XOX10, ZERR13A]

Bruce Morgen. Bruce has been an active ZCPR and Z-System programmer, patcher, and troubleshooter for some time. A tireless beta tester, he's exercised many programs which otherwise might not have been tested. [ANY4 CL05, CMD13, DEV11A, FOR12, GOTO14, HOLDZ11, LGET11, LLF11, MEX+2Z, MU314, NAME10, PAGE21, SDD301P2, SETDZ3, SETFILE11, SHCTRL11, SHSET22, TEX14, TRIM10, VREN10, ZIPDIR12]

Gene Pizzetta. Gene has been incredibly prolific these past two years not only providing new and useful utilities, but improving on others that have needed upgrading for some time. [BAK13, CHKDIR10, CONCAT10, CRCZ11, D17, DATSTP13, DETABZ13, DSKNUM15, ECHO12, ERASE56, LIST10, MDU11, OE15, PRETTY30, PRTASM19, QUIET13A, RCOPY11, RENAME37, REVFN11, ROMAN03A, SAPZ11, TCVIEW24, UNARCZ10, UNJUST12, W23, WHEEL33, XXI10, Z34ERR12]

Terry Hazen. Terry is known for his fine line of system utilities which operate in both ZCPR3 and CP/M environments. [ACOPY30, DD17, ENVSRC10, ERAZ14, PRNTXT15, RENAMZ17, SCAN23A, TCSRC14, UNERAZ12, ZP11]

The First Few Nuggets...

CHKDIR10, SAPZ11 **(ZSUS Vol 1 #12)**

CHKDIR.COM and SAP-Z.COM, introduced this past October by Gene Pizzetta, are really handy utilities for analyzing and ordering disk directories. Based on the last version (1.8) of CLEANDIR, CHKDIR does some helpful diagnostics on a disk directory. It reports: zero-length files, duplicate entries, extents and user areas greater than 31, records greater than 128, illegal filename characters, and duplicate allocation group assignments. CHKDIR does no writing, so poses no danger to your directory or to DateStamper !!!TIME&.DAT files. It calls the error handler on error, so you can abort if calling it from a ZEX or SUB script. Invoking CHKDIR, you might see something like this...

```
A0:BASE>CHKDIR D:
Checking Drive D:  -> Reading ..
 Zero length file       30:-.123
    1 null file(s)
Sorting .. Checking ..
 User Over 31       -> 134:^Z=.^T
 Two files use allocation block 0041  3:RENAME37.LBR
  15:ALIASES.HLP
 Two files use allocation block 0043  3:RENAME37.LBR
  15:ALIASES.HLP
 Two files use allocation block 005D   3:SCAN23A.LBR
  15:ALIASES.HLP
 Two files use allocation block 008D  3:UNERAZ12.FOR
    15:ALIASES.HLP
```

...which is what I was alarmed to see and I'm sure would cause many others some concern. This is just the information, however, you would want to begin setting your directories in order. First, I erased the files with duplicate block assignments on D3: and D15: (I, of course, had these backed up elsewhere) as well as the null-length file which somehow got misdirected to D30:. Then, since I couldn't access the file which was mysteriously assigned to D134:, I pulled out the trusty Z-System DU3 disk utility, located the file in the directory tracks and carefully changed the leading byte to E5 to mark it as erased. Running CHKDIR again, all was now well with disk D:...

```
A0:BASE>CHKDIR D:
Checking Drive D:  -> Reading .. Sorting .. Checking ..
No Problems.
```

Confident that the directories were now garbage-free, I next turned to SAP-Z to sort and pack them. SAP-Z is based on the CP/M program SAP.COM (vs 6.0) and will work only under ZCPR3 and systems with a CP/M 2.2 compatible bios. It will operate on both floppy and hard drives (a TPA of 50k will allow 1500 filenames to be sorted), but should never be used unless you're sure there are no trashed entries in your directory tracks (hence, the importance of using CHKDIR first).

SAP-Z accepts a DU or DIR, but only the drive is significant. If only a user area is specified, the default drive is selected. When only the filename is given, a help message is displayed (for safety purposes, the default DU is not accessed in this case).

A nice feature about SAP-Z is that it will optionally erase all null- length files on a disk, such as temporary files with an .$$$ extension. Knowing that many users want to retain disk-

Continued Page 24

Review of BDS 'Z'

Bringing C and Z-System Together

by Carson Wilson

For over a year now, Sage Microsystems East in cooperation with Leor Zolman has been offering a customized version of Leor's famous BDS 'C' compiler which makes the most of Z System. This new version, christened "BDS 'Z'," sells for between $60 and $90, and generates Z programs at will (those unfamiliar with Z System should see "The Z System Corner" in this and previous issues of *The Computer Journal*). However, perhaps because C is not his native computer language, Jay has been unusually quiet about this package. Until now little information about BDS Z has been available outside of brief promotional notices and an occasional snatch of conversation overheard on Jay's Z-Node #3.

After agonizing for months about BDS's compatibility with other C compilers, the size of the programs it produces, and its speed and ease of operation, I finally broke down and bought a copy of BDS Z from Jay at last April's Trenton Computer Festival. Having used the package for a few months I know that I probably would have "broken down" sooner had I known more about it. Hence this review. I hope it encourages you to try BDS Z.

First and foremost, BDS Z is a mature compiler package, and it shows. The documentation has obviously gone through many, many revisions. The manual anticipates many of the questions a new user is likely to ask, and answers them in an organized fashion. The compiler, linker, and link libraries included with BDS Z also show their maturity in several ways. In my experience the compiler and linker have yet to crash, make good use of memory, and display information clearly (see figure 1). In addition, the large variety of routines included in the link libraries are well thought out and allow a great deal of flexibility of use.

The "feel" of BDS Z is one of being much closer to the compilation process than with Turbo Pascal or other CP/M high level language compilers. For example, the BDS package lets you control the compilation and linkage processes separately, and this has several benefits. First, you can write and compile one part of a program at a time. If part of your code doesn't compile correctly the first time you aren't forced to re-compile the entire program repeatedly as with Turbo Pascal. Rather, with BDS Z you need edit and recompile only the offending function. The separate linker also allows "hackers" to incorporate their own super-efficient assembly language routines into C programs. But let me stress that while BDS Z *allows* you to perform these programming tricks, it doesn't *force* you to. It is also easy to create elegant and useful programs from a single C source code file in one step. With BDS Z there is no one "correct" mode of operation, so the choice between simplicity and complexity can be tailored to meet the task at hand.

The compiler and linker are also very swift, especially with smaller programs. Larger programs do take some time to compile (still, we're usually talking about seconds, not minutes). But since compilation is a two-pass process, error messages are usually quite explicit, and unlike Turbo Pascal (don't get me wrong, Turbo is a fine tool too), the compiler normally doesn't stop at the first error. While this sometimes means a flood of messages result from a single mistake (forgetting to close a C comment will do it), it also means you can reduce overall development time by fixing more than one bug between compiles. BDS Z can also output an error file whenever a compiler error occurs (see figure 2). This file, named PROGERRS.$$$, can in turn automate the edit-compile-debug process to a very high degree. I've even written a Z System Alias that appends the error file to my program source as a comment and automatically returns me to my editing session if errors occur (see figure 3).

```
A>cc bchart.c
BD Software C Compiler (for ZCPR3) vZ2.0  (part I)
  30K elbowroom
BD Software C Compiler vZ2.0 (part II)
  28K to spare
A>clink bchart
BD Software C Linker (for ZCPR3)  vZ2.0

Last code address: 0F7C
Externals start at 0F7D, occupy 0006 bytes, last byte at 0F82
Top of memory: CEFF
Stack space: BF7D
Writing output...
  39K link space remaining
A>
```

Figure 1. Typical Compilation and Linkage Session

That's the Good. And then....

Okay, those are the benefits of BDS Z; how about the drawbacks? There *are* a few. First, I would generally not recommend this package for those just learning to program their computers. The C language is itself a rather formidable barrier, consisting as it does of compact, cryptic-looking statements. And the down-

Carson Wilson has been active in the ZCPR community for over four years now, and has enjoyed every minute. He is author of the Z System Display Editor (ZDE), co-author of Z System Disk Operating System (ZSDOS), and SysOp of Z-Node #11, 312/764-5162, Chicago. In real life Carson is a doctoral student in Political Science at Loyola University of Chicago.

side of the flexibility of BDS Z is that, at least initially, it requires a greater understanding of what's going on beneath the surface than self-contained compiler/editor packages like MBASIC or Turbo Pascal. For example, some of BDS Z's error messages and their explanations in the manual assume that you already know what a "linker" is.

Second, even experienced C programmers are likely to have some difficulty with BDS' exclusion of several elements of the C language standard as specified by Kernighan and Ritchie. Most, perhaps all, of the omissions in the BDS implementation can be worked around, and I don't doubt that each choice was made carefully and with a view to the peculiar needs of the CP/M environment (C was originally implemented on much larger machines). Nonetheless, I find that C source from other compilers can almost never be ported to BDS without some adjustments, and this can become quite a nuisance.

```
#include <stdio.h>
main()
begin
{
        char a;
        byte b;

        for (a = 0; a <= (255-92); ) do
        {
                printf("%03d = %08b  %03d = %08b  %03d = %08b  %03d = %08b  \
%03d = %08b\n", a, a, a+23, a+23, a+46, a+46, a+69, a+69, a+92, a+92);
                if ((++a % 23) == 0)
                        pause();
        }
}
end.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
BCHART.C: 6: Undeclared identifier: byte
BCHART.C: 15: Expecting "while"
BCHART.C: 16: Expecting "("
BCHART.C: 17: Encountered EOF unexpectedly
 (check curly-brace balance)
```

Figure 2. Typical Error Report and Accompanying Source Code.

Another minor drawback: BDS Z emits (and is probably written in) 8080 code only. I gather that this is because BDS C was born when the Z80 chip was still considered an "innovation." While there are certainly arguments in favor of a code that runs on the older equipment, one can't help but imagine the improvements Z80 code could bring to the BDS compiler and its progeny. Perhaps we could even "earn back" some of the C language constructs that have been omitted from BDS Z due to the constraints imposed by earlier microprocessors.

Finally, BDS Z still has some bugs. Notably, though the compiler is supposed to set the Z System error flag when it finds an error in your code, some sorts of errors don't set the flag (this is why my Z System Alias senses the presence of the error file instead, which *is* reliably produced). And a few functions don't operate exactly as described in the manual. For example, the manual says that the "topofmem" function "returns a pointer to the last byte of user memory," but I found by trial and error that it does *not* work unless the program calling it was linked using the "-t" or "-n" options. Still, there is so much versatility built into the standard functions of BDS Z that a workaround is usually available. I easily solved my problem with "topofmem" by substituting another of BDS' standard routines. Another difficulty with the 'Z' version is that the debugger and RED editor that come with BDS C haven't yet been updated to work with the newer compiler (they still work with the standard compiler included in the $90 package, however).

Complaints aside though, my overall impression of BDS Z is quite favorable. This is a very fast C compiler/linker package, far faster than others on the market. It allows you to generate efficient, useful CP/M and Z System programs. BDS Z's small runtime library lets you write utility programs that are considerably less bulky than those generated by other high level language compilers (some as tiny as 3k!). The BDS compiler and linker include command line options for further optimization. For example, on many CP/M systems program code size can be reduced by grabbing one or more of the Z80's restart vectors. While BDS Z achieves some of this performance by implementing only a subset of the C language, the manual is quite explicit about the limitations.

The BDS compiler has also been around long enough to have generated all kinds of support packages which make up for its weaknesses by extending the language and its environment. One very significant extension of BDS for Z System users is Cameron Cotrill's Z System interface project. Cameron's library of C routines allows BDS programs to make Z System environment calls similar to those offered by the VLIB and Z3LIB assembly language libraries. Using this package (available on bulletin boards as CFORZxx.LBR) it is easy to write full Z System utilities that access such structures such as named directories, the shell stack, and the error flag for enhanced performance. You can even access the new TCAP's business graphics character set from within your C programs!

```
xif
zde $1.c /n
cc $1.c $2
if ex progerrs.$$$$$$
   echo fix
   if in
        concat $1.c = stcmt progerrs.$$$$$$ endcmt /a
        era progerrs.$$$$$$
        c $*
        else
        era progerrs.$$$$$$
   fi
   else
   cl $1
   if er
        c $*
   fi
fi
```

Figure 3. Z System Alias for Automated Debugging

To sum up, there is really no limit to what you can do with BDS Z. This can be a real thrill at times, but it can also be dangerous. Because there is so much flexibility built into the package, I sometimes have trouble concentrating on a

Continued Page 36

PMATE / ZMATE Macros

Introducing a New Column

By Clif Kinne

That pillar of support to *TCJ*, your friend and mine, Jay Sage, feels that enough of our readers are current, or prospective, users of the PMATE editor to warrant a regular PMATE column. Being loathe to curtail any of his other endeavors to get such a column started, he invited me to take it on.

PMATE, as many of you must know by now, is Jay's favorite editor. It was also, and probably still is, the favorite of Ward Christensen, that venerable guru of hackers everywhere, originator of the bulletin board concept and author of the first BBS program (CBBS) and the first telecommunications program (MODEM). That I, too, have been crazy about PMATE for almost ten years adds nothing to the testimonial, but it may explain why Jay thought I might do justice to a column on it.

Briefly, PMATE is an extremely customizable and configurable editor, with an outstanding built-in macro language for text processing. It is available in CP/M and MS-DOS versions, and now has been rewritten by Jay and Bridger Mitchell in a new version, ZMATE, to take full advantage of the Z-System. Jay's dissertation on ZMATE, which began in *TCJ* #46, promises to be a better in-depth description of PMATE than any of the previously published PMATE manuals. All of you PMATE users will want to read The Z-System Corner to benefit from Jay's insight into PMATE. I shall endeavor not to be redundant.

After nearly 10 years, both Jay and I find that we are still discovering new wrinkles in and uses for PMATE macros. It is primarily for that reason that we feel that a column should be reinstituted. I say reinstituted because there was a "Macros of the Month" column, edited by Michael Olfe, in *Lifelines* magazine in 1981 and 1982. That column gave many of us a jump start at taking advantage of the macro language.

Maybe we can jump start those who have acquired PMATE more recently and let this column be a forum where all of us can exchange macros and other ideas to further expand PMATE's utility. So let us hear from you with a pet macro, a suggestion, questions, or even just a vote to continue the column.

My thinking at the moment is that the main focus in each issue will be on one or two major macros submitted by readers or culled from Jay's collection or mine. These will include many of the most useful macros from the *Lifelines* column.

In addition, there are some philosophical points of view, guidelines to setting up your macro area, pitfalls, etc., that have filtered up over the years. We shall offer comments and general observations on such matters for your consideration.

This Month's Macro

To get our feet wet gradually (mine as well as yours), I have chosen a rather modest macro that is not only a useful subroutine in its own right, but illustrates a couple of techniques that will be very useful in many other macros:

1. It calls a subroutine of its own.
2. It builds a macro in a buffer and then executes it.

If you write in standard programming languages such as BASIC or C, you are well acquainted with subroutines, but have you ever seen routines that, themselves, write other routines and then execute them?

The main macro will read the first string of decimal digits following the cursor and store it as an integer in V1 (variable 1). This macro is a variation on one written by Ward Christensen and published in *Lifelines* in 1982. He gave it the name 'D' for "Decimal", and we shall use that name here.

It is useful when you want a macro to arithmetically modify numbers in your text, want the user to supply numeric input to initialize a variable, etc. We will give examples of such uses in future issues of *TCJ*.

A complete specification for this macro is given with the source code in Listing 2.

The Auxiliary Macro, Control-D

To the macro that 'D' calls as a subroutine I have arbitrarily given the name control-D or '^D'. It simply checks whether the character under the cursor is a digit. It is 14 bytes long and is called twice by 'D', resulting in a net saving of 10 bytes. Furthermore, having it in your permanent macro area will encourage you to use it in one-shot macros, where you otherwise might not bother.

The complete specification for this macro is given in Listing 1.

Variations

There are quite a few variations of this macro that might suit your needs. First, if you don't mind PMATE's own error control, you could save some space in your macro area by omitting lines 1 - 5 in Listing 2, which abort the

Clif Kinne is a retired computer designer. He cut his teeth on vacuum tube and acoustic delay line machines in the fifties, made the transition to transistors and magnetic cores in the sixties, left the field to his children in the seventies, and tried, vainly, to catch back up with them in the eighties. He can be reached by voice at 617-444-9055, or via a message on Jay's BBS, 617-965-7259. His address is 159 Dedham Ave., Needham, MA 02192

Listing 1. Auxiliary macro, '^D'

```
;       FUNCTIONAL SPECIFICATION
;
;       Pushes -1 on the stack if cursor is on a digit.
;       Pushes 0 on the stack if it is not.
;
;       VARIABLE USAGE    - None
;       BUFFER USAGE      - None
;       SUBROUTINES USED  - None
;       SIDE EFFECTS      - None
;
;       USAGE
;
;       .^D@S{Code to be executed if you're on a
;       digit}
;                                 ;or
;       .^D@S'{Code to be executed if you're on a
;       non-digit}
;
;       Note that ASCII "/" is 1 less than ASCII "0"
;       and ASCII ":" is 1 more than ASCII "9"
;
@T>"/   ;IF the character at the cursor is greater than "/"
&       ;   and
(@T<":) ;   also less than ":"
,       ;THEN push the Boolean result (-1 if true,
;          0 if false) onto the stack.
;
;       Note also that parentheses are not required around
;          the first conjunct (or disjunct) of a series.
```

macro if it is called from buffer 0.

Second, you will note that 'D' is limited to integers in the range +/-32767. This could easily be changed, of course, to 0...65535 by changing line 6 of Listing 2 from Q-to 0Q-.

Third, you can reverse the effect of a leading argument on the macro call, 0.D, by changing line 7 of Listing 2 from '@A{0V1}' to '@A=0{0V1}'. Then .D will add to V1, while 0.D will replace the contents.

Finally, it is possible to eliminate the side effects described in Listing 2. First I must digress somewhat.

Version 1, as given in Listing 2, is pretty much what I have used for 6 or 8 years. I put the digit test code into a subroutine fairly recently, but I never gave much thought to side effects, just lived with them, I guess. However, when I first passed a draft of this column to Jay, he immediately spotted the side effects, and said he has been frustrated by side effects in other macros, though he has not had an equivalent of this one.

Fortunately, as Jay pointed out, for MS-DOS users, and now for ZMATE also, there is a new primitive command, B@SE, that makes avoiding these side effects relatively easy. For users of the original CP/M PMATE, one can get around the side effects, but the required techniques are much more complex. However, as I mentioned, I found the side effects a minor annoyance at worst, and you may too.

The use of the B@SE command is best explained by example. Say you are working in buffer 5. The following se-

Listing 2. Main macro, .D, version 1.

```
;
;                   FUNCTIONAL SPECIFICATION
;
;1. Can be called from any buffer except buffer 0.
;2. Stores the 1st integer to the right of the cursor in V1.
;3. If a zero argument preceeds the macro call (0.D) the
;     integer will add to, not replace, V1's contents.
;4. Carries a leading minus sign into V1.
;5. Handles numbers from -32767 to +32767
;6. When called, requires the cursor to be on, or to the
;     left of, the number's first digit.
;7. Terminates with the cursor just past the final digit
;
;                         VARIABLE USAGE
;
;   V1  Holds the signed integer at the end of the macro.
;
;                         BUFFER USAGE
;
;   Buffer 0 is used to build the macro, nVA1.
;
;                       SUBROUTINES USED
;
;   .^D  Tests whether the character under the cursor is a
:          decimal digit, "0".."9".
;           Pushes -1 on the stack if it is.
;           Pushes 0 on the stack if it is not.
;
;                         SIDE EFFECTS
;
;1. Wipes out contents of buffer 0.
;2. If the macro which called 'D' was using the Tag that use
;     will be destroyed by this macro.

;    Abort the macro if you are in buffer 0:
;
@B=1    ;IF you are in Buffer 0,                          1
{       ;THEN                                             2
  GCannot call this macro from buffer 0$  ;Remind user.   3
  %     ;Abort macro.                                     4
}       ;END IF                                           5
;    End Abort if Buffer 0

Q-      ;Display numbers as +/-, not as positive only.    6
@A{0V1} ;If the calling arg. was 0, do not clear V1.      7
$       ;(Character to separate '}' from '[' so that      8
;          "}[" isn't interpreted as an "else")

;    Search for the first digit to the right:
[       ;Start loop.                                      9
  @T=0% ;If at End-of-File, terminate macro.              10
  .^D   ;Is cursor on a digit?                            11
  @S    ;If it is,                                        12
  _     ;escape loop.                                     13
  M     ;Else move right                                  14
]       ;and loop.                                        15
;    End search.  You are on the first digit (or at end
;                 of buffer).

@C>0    ;IF first digit is not on column 0 of line 0      16
{       ;THEN                                             17
;    Pick up minus sign, if any:
  T     ;Tag first digit.                                 18
  -M    ;Move left.                                       19
  @T="-;If preceding column is a - sign,                  20
  {T}   ;   retag.                                       21
;    Got minus sign.
}       ;END "IF first digit..."                          22

$       ;(Character to separate } from [                  23

;    Search for the next non-digit:
[       ;Start loop                                       24
  M     ;Move right.                                      25
  .^D   ;Is cursor on a digit?                            26
  @S'   ;If not, @S=0, @S'=1, and you will escape loop.   27
]       ;Else loop again.                                 28
;    End search.  You are just past the last digit.

#BC     ;Copy the signed string of digits, to buffer 0.   29
T       ;Tag.                                             30
IVA1%$  ;Insert the string, "VA1%", at the cursor.        31
#BN     ;Append "VA1%" to buffer 0, removing it from      32
;          the current buffer.
.0      ;Execute buffer 0, nVA1, adding n to V1.          33
```

quence is possible:

```
...
@B,     ;Push the current buffer no. (5) onto the
        ;stack. (actually, @B returns 0 for the T
        ;buffer and one plus the buffer number for
        ;all others, but B@SE takes care of this)
B2E     ;Move to buffer 2.
...     ;Execute some code on the contents of
        ;buffer 2.
B@SE    ;Return to buffer 5.
...     ;Continue work in buffer 5....
```

Furthermore, this technique also works if @B is put into one of the variables, V0 - V9, instead of onto the stack. This is advantageous if you have to go back and forth between two buffers several times. It saves repeatedly pushing @B onto the stack each time. This variation is applicable here. I have arbitrarily used V7, and to avoid another side effect, I push @7 onto the stack before use and pop it back into V7 after I'm through.

Listing 3 is source code with which I propose to replace lines 16–33 of Listing 2. As you can see, this will leave buffer 0 restored to its condition before the call to 'D', except that its cursor will be at the top of the buffer, whether it was before or not. With some effort, even this change could be prevented, but the main reason for leaving buffer 0 intact is in case cut-and-paste operations were in progress when 'D' was used, and they do not depend on the cursor position. And the tag command, T, is no longer used, leaving the tag status undisturbed.

For Next Time

Among topics I hope to discuss while awaiting the deluge of responses from you other readers are:

1. CP/M vs MS-DOS. Considerations in adapting macros written for one version of PMATE for use in the other.
2. Small, general purpose, utility macros, which will greatly simplify our writing and discussion of larger, more specialized macros later on.
3. The organization and management of the permanent macro area.•

Listing 3. Main macro, 'D', version 2. Source code changes for MS-DOS and ZMATE macros to eliminate side effects.

```
;                       VARIABLE USAGE

;   V1    Holds the signed integer at the end of the macro.
;   V7    Stores the ID number of the working buffer.

@C>0    ;IF first digit is not on column 0 of line 0      16
{       ;THEN                                             17
;    Pick up minus sign, if any:
  -M    ;Move left.                                       18
  @T="-';If preceding column is not a - sign,             19
  {M}   ;Move right again.                                20
;    Got minus sign.
}       ;END "IF first digit..."                          21

;    Prepare buffer 0 to receive macro:
@7,     ;Save caller's V7 on the stack.                   22
@BV7    ;Put ID of current buffer in V7                   23
BE      ;Go to buffer 0.                                  24
A       ;To top.                                          25
IVA1%$  ;Insert the string, VA1%, at top of B0            26
A       ;Leave cursor at start of B0.                     27
B@7E    ;Return to your calling buffer (This works only  28
;          with MS-DOS and ZMATE).

;    Transfer the integer one digit at a time:
[       ;Start loop.                                      29
  @T,   ;Push minus sign or ASCII digit on the stack.     30
  BE    ;Go to buffer 0.                                  31
  @S    ;Pop minus sign or ASCII digit off the stack.     32
  I     ;Insert into buffer 0.                            33
  B@7E  ;Return to calling buffer for next digit.         34
  M     ;Move right                                       35
  .^D   ;Is next character a digit?                       36
  @S'   ;If yes, @S=1, (so @S'=0 and you will loop);      37
]       ;   else @S'=1 and you will escape loop.         38

;    Transfer complete.  Cursor is just past last digit.
.0      ;Execute buffer 0, nVA1, adding n to V1.          39

;    Restore buffer 0:
BE      ;Go to buffer 0.                                  40
S%$     ;Move to end of macro.                            41
0K      ;Delete line left, removing macro.                42
B@7E    ;Return to your calling buffer.                   43
@SV7    ;Restore caller's V7.                             44
```

Continued from Page 33

single task. Other high-level CP/M languages such as Turbo Pascal and MBASIC generally insulate me from the operating system, reducing the number of decisions involved in performing a given task. BDS Z (and C in general), by constantly enticing me to go one step deeper, places greater demands on my skills and knowledge. This can be quite rewarding when it leads to a solution, because knowing that I have studied a problem in depth I am usually more confident of my solution. However, the temptation to explore can also distract unnecessarily from the task at hand if I am not careful.

Of course, there is much more to tell about BDS Z that this space will hold. There are literally hundreds of programs written for BDS C whose source code is available through *The C User's Group* (CUG). As I mentioned above, a full-screen editor which uses BDS's extended error reporting to form an efficient development environment comes with the full-priced BDS package, as does a debugger that displays the symbols in your C source code as you step through a program (both of these are described in detail in the manual). And the compiler and linker boast many command line options that I haven't touched upon. To learn more, contact me at Antelope Freeway or Jay Sage at the addresses given in *The Z System Corner*. Or boldly order a copy and see ("C?" "Z"?) for yourself!•

Continued from Page 6

The result is a binary fraction 0.00001001_b or 0.09_h. We can still use the bits shifted out to increase accuracy. Pad 10101 with 0's to obtain an 8-bit value 10101000_b or $A8_h$. The result then becomes $0.09A8_h$ 0.0377_d. What we have done is reversed our original division to 7 divided by $B9_h$ and $7/185_d$ = 0.0378_d, which brings us to the next part.

Other Things to Do: Inversion, Exponentiation & Roots

As we just saw, by negating the log of a value, and then taking the antilog we actually performed an inversion or 1/value. But there are other math functions that can be achieved--such as exponentiation. Just multiply the log with the exponent. For example 7^4 is calculated as $LOG_2 7 * 4$ or $2.CE_h * 4$ which gives $B.38_h$. Reintroduce the error to $B.2A_h$ and reverse shift to obtain 950_h or 2384_d. It should of course be 2401, an error of 0.7%. When multiplying logs, the error also multiplies.

Roots can be extracted with somewhat greater precision by dividing the log by the root. When a log is divided, the error is reduced accordingly. $\sqrt[2]{B9_h}$ would be $LOG_2 B9_h/2$ or $7.88_h/2$ giving $3.C4_h$. Reintroduce the error to $3.B4_h$ and do the reverse shift. The result is $0D.A_h$ $\sqrt[2]{B9_h}$=$0D.99FC_h$.

Tweaking the Code

There are numerous tricks to improve performance and reduce space. Here are a few points:

(1) **IMPORTANT!** When one of the arguments is a constant, always code the log of that constant into the program to save one conversion routine.

(2) When you have an idea of the range of an argument, use a separate shift routine for it, starting off with the highest bit that is likely to be set i.e. if the high byte in a 16-bit argument is 00, proceed as if for 8 bits. Use byte or nibble shifts where possible.

(3)To save space in longer error tables, one byte in an entry can actually hold an 11-bit error if left shifted 3 times. The 3 MS bits in the error are always 000. For example, the error of 0.91_h is $0.14C_h$ but can be stored as $A6_h$. It is not worth to have 256 entries in an 8-bit table, but 11 bits can produce good results.

In Conclusion

And that's all there is to it. As far as speed improvement, an 8-bit division executes slightly faster using logs, but when we get up to 16-bit or larger arguments (especially if one is a constant) the savings are much more impressive. There is however a great deal of difference between various controller families, and it might pay off to check execution times before committing the code.

In a recent exercise with an 8048 controller I had to divide a 19 bit constant ($2\ 000\ 000_d$) by a 16 bit variable within a wide range. To try various routines I selected a value of 2000_d as the variable, for the obvious result of 1000_d. These aren't nice and round figures in hex. Using a 24-bit standard division routine took 680 MC's (Machine Cycles) and produced the expected 1000. Reducing accuracy by removing 5 bits and then using a 16-bit routine took 430 MC's with a result of 1008. The final solution was to use logs, with a 128 byte error table, which took only 140 MC's and returned 1000 as the result. The penalty was code size. The 24-bit division routine used 48 bytes while the log routine occupied 68 plus another 128 for the error table.

Happy logging. •

Continued from Page 40

option does not pay off. The older more tried and true method was the best option.

OF LANS

The major drive in the PC work of late, is porting our products to LAN based platforms. What has been interesting is finding out how the different LAN companies do their interfaces. The standard interface for programs to use the LAN are all based on NETBIOS. NETBIOS is suppose to be a standard interface which programmers can write applications to use. By using it, supposedly any program then can access the LAN and send data to other users. As with other things in the industry, the standard is rather vague and we have had many problems. We started with IBM LAN product, went to 3COM, and are starting on NOVEL next. Each loads the NETBIOS differently and NOVEL gives you the option altogether.

Our program is a device driver and with IBM their NETBIOS is also a device driver. We loaded our program after IBM's and all worked just fine. 3COM's NBP (NetBios Program) is loaded as a TSR and therefore was not installed when we started checking for LAN activity. I believe Novel's NETBIOS is also a TSR and we expect the same problems as with 3COM's. Our first solution is to make sure your program can continue to work until the NETBIOS is loaded. Typically that means testing, waiting, testing, then trying again.

The major problem was using 3COM and interrupts. We redirect the keyboard interrupt handler to our own. The adapter board also use the COM3 interrupt which we find getting turned back off. Apparently one of 3COM's programs turns off the PIC (interrupt controller) on us. We have been using Token Ring cards and when we tested Ether net cards found a problem with 3COM not redirecting the timer tick correctly. We also found IBM doing the same on one of their LAN programs. It seems these big companies do not follow their own instructions when it comes to interrupt processing. It turned out the only way to handle the stolen interrupts was give our interrupt driver a GO command. I now have the program delay setting interrupts until it receives the GO command. At that point it then installs our interrupt redirection and everything works fine. Putting the GO command last in the AUTOEXEC.BAT now solves all the problems.

We have seen some other minor problems, things like excessive delays, time outs, lost names, and other problems that come and go, but for the most part it works in our minor test. We have yet to put more than 6 units on line. We have a test coming up soon with over 100 units and will report later on those results. Overall the use of LANs looks good, but I still have lots of reservations. I personally feel that lots of expensive LANs are in places where cheaper and less maintenance intensive option are possible. One of our big problems is system supervision, someone has to be in charge of the LAN system and all the security, and software problems that entails. Most small organizations have neither the resources or experienced personnel to do the supervision properly.

That's ALL

Well this is a busy month, so off to other things.

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Special Close Out Sale on these back issues only.

3 or more, $1.50 each postpaid in the US or $2.50 postpaid surface outside US.

Issue Number 1:

- RS-232 Interface Part 1
- Telecomputing with the Apple II
- Beginner's Column: Getting Started
- Build an "Epram"

Issue Number 2:

- File Transfer Programs for CP/M
- RS-232 Interface Part 2
- Build Hardware Print Spooler Part 1
- Review of Floppy Disk Formats
- Sending Morse Code with an Apple II
- Beginner's Column: Basic Concepts and Formulas

Issue Number 3:

- Add an 8087 Math Chip to Your Dual Processor Board
- Build an A/D Converter for Apple II
- Modems for Micros
- The CP/M Operating System
- Build Hardware Print Spooler: Part 2

Issue Number 4:

- Optronics, Part 1: Detecting, Generating and Using Light in Electronics
- Multi-User: An Introduction
- Making the CP/M User Function More Useful
- Build Hardware Print Spooler: Part 3
- Beginner's Column: Power Supply Design

Issue Number 8:

- Build VIC-20 EPROM Programmer
- Multi-User: CP/Net
- Build High Resolution S-100 Graphics Board, Part 3
- System Integration, Part 3: CP/M 3.0
- Linear Optimization with Micros

Issue Number 18:

- Parallel Interface for Apple II Game Port
- The Hacker's MAC: A Letter from Lee Felsenstein
- S-100 Graphics Screen Dump
- The LS-100 Disk Simulator Kit
- BASE: Part Six
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 1

Issue Number 19:

- Using the Extensibility of Forth
- Extended CBIOS
- A $500 Superbrain Computer
- BASE: Part 7
- Interfacing Tips & Troubles: Communicating with Telephone Tone Control, Part 2
- Multitasking & Windows with CP/M: A Review of MTBASIC

Issue Number 20:

- Designing an 8035 SBC
- Using Apple Graphics from CP/M: Turbo Pascal Controls Apple Graphics
- Soldering & Other Strange Tales
- Build an S-100 Floppy Disk Controller: WD2797 Controller for CP/M 68K

Issue Number 21:

- Extending Turbo Pascal: Customize with Procedures & Functions
- Unsoldering: The Arcane Art
- Analog Data Acquisition & Control: Connecting Your Computer to the Real World
- Programming the 8035 SBC

Issue Number 22:

- NEW-DOS: Write Your Own Operating System
- Variability in the BDS C Standard Library
- The SCSI Interface: Introductory Column
- Using Turbo Pascal ISAM Files
- The Ampro Little Board Column

Issue Number 23:

- C Column: Flow Control & Program Structure
- The Z Column: Getting Started with Directories & User Areas
- The SCSI Interface: Introduction to SCSI
- NEW-DOS: The Console Command Processor
- Editing the CP/M Operating System
- INDEXER: Turbo Pascal Program to Create an Index
- The Ampro Little Board Column

Issue Number 24:

- Selecting & Building a System
- The SCSI Interface: SCSI Command Protocol
- Introduction to Assemble Code for CP/M
- The C Column: Software Text Filters
- Ampro 186 Column: Installing MS-DOS Software
- The Z-Column
- NEW-DOS: The CCP Internal Commands
- ZTime-1: A Real Time Clock for the Ampro Z-80 Little Board

Issue Number 25:

- Repairing & Modifying Printed Circuits
- Z-Com vs. Hacker Version of Z-System
- Exploring Single Linked Lists in C
- Adding Serial Port to Ampro LB
- Building a SCSI Adapter
- NEW-DOS: CCP Internal Commands
- Ampro 186 Networking with SuperDUO
- ZSIG Column

Issue Number 26:

- Bus Systems: Selecting a System Bus
- Using the SB180 Real Time Clock
- The SCSI Interface: Software for the SCSI Adapter
- Inside Ampro Computers
- NEW-DOS: The CCP Commands (continued)
- ZSIG Corner
- Affordable C Compilers
- Concurrent Multitasking: A Review of DoubleDOS

Issue Number 27:

- 68000 TinyGiant: Hawthorne's Low Cost 16-bit SBC and Operating System
- The Art of Source Code Generation: Disassembling Z-80 Software
- Feedback Control System Analysis: Using Root Locus Analysis & Feedback Loop Compensation
- The C Column: A Graphics Primitive Package
- The Hitachi HD64180: New Life for 8-bit Systems
- ZSIG Corner: Command Line Generators and Aliases
- A Tutor Program in Forth: Writing a Forth Tutor in Forth
- Disk Parameters: Modifying the CP/M Disk Parameter Block for Foreign Disk Formats

Issue Number 28:

- Starting Your Own BBS
- Build an A/D Converter for the Ampro Little Board
- HD64180: Setting the Wait States & RAM Refresh using PRT & DMA
- Using SCSI for Real Time Control
- Open Letter to STD Bus Manufacturers
- Patching Turbo Pascal
- Choosing a Language for Machine Control

Issue Number 29:

- Better Software Filter Design
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 1
- Using the Hitachi hd64180: Embedded Processor Design
- 68000: Why use a new OS and the 68000?
- Detecting the 8087 Math Chip
- Floppy Disk Track Structure
- The ZCPR3 Corner

Issue Number 30:

- Double Density Floppy Controller
- ZCPR3 IOP for the Ampro Little Board
- 3200 Hackers' Language
- MDISK: Adding a 1 Meg RAM Disk to Ampro Little Board, Part 2
- Non-Preemptive Multitasking
- Software Timers for the 68000
- Lilliput Z-Node
- The ZCPR3 Corner
- The CP/M Corner

Issue Number 31:

- Using SCSI for Generalized I/O
- Communicating with Floppy Disks: Disk Parameters & their variations
- XBIOS: A Replacement BIOS for the SB180
- K-OS ONE and the SAGE: Demystifying Operating Systems
- Remote: Designing a Remote System Program
- The ZCPR3 Corner: ARUNZ Documentation

Issue Number 32:

- Language Development: Automatic Generation of Parsers for Interactive Systems
- Designing Operating Systems: A ROM based OS for the Z81
- Advanced CP/M: Boosting Performance
- Systematic Elimination of MS-DOS Files: Part 1, Deleting Root Directories & an In-Depth Look at the FCB
- WordStar 4.0 on Generic MS-DOS Systems: Patching for ASCII Terminal Based Systems
- K-OS ONE and the SAGE: System Layout and Hardware Configuration
- The ZCPR3 Corner: NZCOM and ZCPR34

Issue Number 33:

- Data File Conversion: Writing a Filter to Convert Foreign File Formats
- Advanced CP/M: ZCPR3PLUS & How to Write Self Relocating Code
- DataBase: The First in a Series on Data Bases and Information Processing
- SCSI for the S-100 Bus: Another Example of SCSI's Versatility
- A Mouse on any Hardware: Implementing the Mouse on a Z80 System
- Systematic Elimination of MS-DOS Files: Part 2, Subdirectories & Extended DOS Services
- ZCPR3 Corner: ARUNZ Shells & Patching WordStar 4.0

Issue Number 34:

- Developing a File Encryption System.
- Database: A continuation of the data base primer series.
- A Simple Multitasking Executive: Designing an embedded controller multitasking executive.
- ZCPR3: Relocatable code, PRL files, ZCPR34, and Type 4 programs.
- New Microcontrollers Have Smarts: Chips with BASIC or Forth in ROM are easy to program.
- Advanced CP/M: Operating system extensions to BDOS and BIOS, RSXs for CP/M 2.2.
- Macintosh Data File Conversion in Turbo Pascal.
- The Computer Corner

Issue Number 35:

- All This & Modula-2: A Pascal-like alternative with scope and parameter passing.
- A Short Course in Source Code Generation: Disassembling 8088 software to produce modifiable assem. source code.
- Real Computing: The NS32032.
- S-100: EPROM Burner project for S-100 hardware hackers.
- Advanced CP/M: An up-to-date DOS, plus details on file structure and formats.
- REL-Style Assembly Language for CP/M and Z-System. Part 1: Selecting your assembler, linker and debugger.
- The Computer Corner

Issue Number 36:

- Information Engineering: Introduction.
- Modula-2: A list of reference books.
- Temperature Measurement & Control: Agricultural computer application.
- ZCPR3 Corner: Z-Nodes, Z-Plan, Amstrand computer, and ZFILE.
- Real Computing: NS32032 hardware for experimenter, CPUs in series, software options.
- SPRINT: A review.
- REL-Style Assembly Language for CP/M & ZSystems, part 2.
- Advanced CP/M: Environmental programming.
- The Computer Corner.

Issue Number 37:

- C Pointers, Arrays & Structures Made Easier: Part 1, Pointers.
- ZCPR3 Corner: Z-Nodes, patching for NZCOM, ZFILER.
- Information Engineering: Basic Concepts: fields, field definition, client worksheets.
- Shells: Using ZCPR3 named shell variables to store date variables.
- Resident Programs: A detailed look at TSRs & how they can lead to chaos.
- Advanced CP/M: Raw and cooked console I/O.
- Real Computing: The NS 32000.
- ZSDOS: Anatomy of an Operating System: Part 1.
- The Computer Corner.

Issue Number 38:

- C Math: Handling Dollars and Cents With C.
- Advanced CP/M: Batch Processing and a New ZEX.
- C Pointers, Arrays & Structures Made Easier: Part 2, Arrays.
- The Z-System Corner: Shells and ZEX, new Z-Node Central, system security under Z-Systems.
- Information Engineering: The portable Information Age.
- Computer Aided Publishing: Introduction to publishing and Desk Top Publishing.
- Shells: ZEX and hard disk backups.
- Real Computing: The National Semiconductor NS320XX.
- ZSDOS: Anatomy of an Operating System, Part 2.

The Computer Journal

Back Issues

Sales limited to supplies in stock.

Issue Number 39:

- Programming for Performance: Assembly Language techniques.
- Computer Aided Publishing: The Hewlett Packard LaserJet.
- The Z-System Corner: System enhancements with NZCOM.
- Generating LaserJet Fonts: A review of Digi-Fonts.
- Advanced CP/M: Making old programs Z-System aware.
- C Pointers, Arrays & Structures Made Easier: Part 3: Structures.
- Shells: Using ARUNZ alias with ZCAL.
- Real Computing: The National Semiconductor NS320XX.
- The Computer Corner.

Issue Number 40:

- Programming the LaserJet: Using the escape codes.
- Beginning Forth Column: Introduction.
- Advanced Forth Column: Variant Records and Modules.
- LINKPRL: Generating the bit maps for PRL files from a REL file.
- WordTech's dBXL: Writing your own custom designed business program.
- Advanced CP/M: ZEX 5.0•The machine and the language.
- Programming for Performance: Assembly language techniques.
- Programming Input/Output With C: Keyboard and screen functions.
- The Z-System Corner: Remote access systems and BDS C.
- Real Computing: The NS320XX
- The Computer Corner.

Issue Number 41:

- Forth Column: ADTs, Object Oriented Concepts.
- Improving the Ampro LB: Overcoming the 88Mb hard drive limit.
- How to add Data Structures in Forth
- Advanced CP/M: CP/M is hacker's haven, and Z-System Command Scheduler.
- The Z-System Corner: Extended Multiple Command Line, and aliases.
- Programming disk and printer functions with C.
- LINKPRL: Making RSXes easy.
- SCOPY: Copying a series of unrelated files.
- The Computer Corner.

Issue Number 42:

- Dynamic Memory Allocation: Allocating memory at runtime with examples in Forth.
- Using BYE with NZCOM.
- C and the MS-DOS Screen Character Attributes.
- Forth Column: Lists and object oriented Forth.
- The Z-System Corner: Genie, BDS Z and Z-System Fundamentals.
- 68705 Embedded Controller Application: An example of a single-chip microcontroller application.
- Advanced CP/M: PluPerfect Writer and using BDS C with REL files.
- Real Computing: The NS 32000.
- The Computer Corner

Issue Number 43:

- Standardize Your Floppy Disk Drives.
- A New History Shell for ZSystem.
- Heath's HDOS, Then and Now.
- The ZSystem Corner: Software update service, and customizing NZCOM.
- Graphics Programming With C: Graphics routines for the IBM PC, and the Turbo C graphics library.
- Lazy Evaluation: End the evaluation as soon as the result is known.
- S-100: There's still life in the old bus.
- Advanced CP/M: Passing parameters, and complex error recovery.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 44:

- Animation with Turbo C Part 1: The Basic Tools.
- Multitasking in Forth: New Micros F68FC11 and Max Forth.
- Mysteries of PC Floppy Disks Revealed: FM, MFM, and the twisted cable.
- DosDisk: MS-DOS disk format emulator for CP/M.
- Advanced CP/M: ZMATE and using lookup and dispatch for passing parameters.
- Real Computing: The NS32000.
- Forth Column: Handling Strings.
- Z-System Corner: MEX and telecommunications.
- The Computer Corner

Issue Number 45:

- Embedded Systems for the Tenderfoot: Getting started with the 8031.
- The Z-System Corner: Using scripts with MEX.
- The Z-System and Turbo Pascal: Patching TURBO.COM to access the Z-System.
- Embedded Applications: Designing a Z80 RS-232 communications gateway, part 1.
- Advanced CP/M: String searches and tuning Jetfind.
- Animation with Turbo C: Part 2, screen interactions.
- Real Computing: The NS32000.
- The Computer Corner.

Issue Number 46:

- Build a Long Distance Printer Driver.
- Using the 8031's built-in UART for serial communications.
- Foundational Modules in Modula 2.
- The Z-System Corner: Patching The Word Plus spell checker, and the ZMATE macro text editor.
- Animation with Turbo C: Text in the graphics mode.
- Z80 Communications Gateway: Prototyping, Counter/Timers, and using the Z80 CTC.

Issue Number 47:

- Controlling Stepper Motors with the 68HC11F
- Z-System Corner: ZMATE Macro Language
- Using 8031 Interrupts
- T-1: What it is & Why You Need to Know
- ZCPR3 & Modula, Too
- Tips on Using LCDs: Interfacing to the 68HC705
- Real Computing: Debugging, NS32 Multitasking & Distributed Systems
- Long Distance Printer Driver: correction
- ROBO-SOG 90
- The Computer Corner

Subscriptions	**U.S.**	**Foreign (Surface)**	**Total**
1year (6 issues)	$18.00	$24.00	
2 years (12 issues)	$32.00	$46.00	
Air Mail rates on request.			
Back Issues			
16 thru #43	$3.50 ea.	$4.50 ea.	
6 or more	$3.00 ea.	$4.00 ea.	
#44 and up	$4.50 ea.	$5.50 ea.	
6 or more	$4.00 ea.	$5.00 ea.	

Issue #s ordered_____

Subscription Total _____

Back Issues Total _____

Total Enclosed _____

All funds must be in U.S. dollars on a U.S. bank

Name_____

Address_____

_____

_____

☐Check ☐VISA ☐MasterCard Exp. Date_____

Card #_____

Signature_____

The Computer Journal
P.O. Box 12, S. Plainfield, NJ 07080-0012
Phone (908) 755-6186

The Computer Corner

By Bill Kibler

It is PC strikes again time. Been working on PC machines lately so have a number of facts, problems, and ideas to pass along.

80287

For some time now I have been trying to buy a 80287 co-processor for my AT. The 12 MHZ version normally runs over $200 and so finds its way to the bottom of the "need–to-buy" list. AMD came to the rescue with a CMOS version for $99 dollars. I saw the stories about the Intel/AMD law suits over it, and decided I had better buy one while the option was available.

AMD ran a few ads and articles, which supplied an 800 number (1-800-888-5590). I called them some time earlier and was promised a 3 week shipment. I think that actually turned into over 2 months, but actually I lost track because it took so long. What I got was the chip, a disk, and installers guide. Chip went in easily and seems to run considerably cooler than the other chips. I have noticed in the past that the 286 chip set runs extremely hot, especially at 12 MHZ. The 287 from AMD is CMOS which runs cooler and faster. Although I told the order taker I had a 12 MHZ machine, the part has no indication of speed.

Speaking of speed, the few benchmarks I have run make considerable improvements. As I continue to test, it amazes me how many programs seem to run faster, even the ones that don't use it. When you start paying attention to speed of program execution, it becomes a rather funny situation. I find programs I thought went slow, aren't really that slow. Graphic programs however are definitely faster by many times. Where you could watch the program fill the screen, you now almost instantly (well close to that) get a full graphic display. We even noticed an increase in the old Windows 2 package, and will be testing Windows 3 later. The disk contain VGA and EGA fractal programs and my test machine only has a mono card in it. The fractal test will have to wait till later, although it is rather a poor choice of programs to not support more formats.

The bottom line is, if you have been waiting to get an co-processor, now seems to be the time to do it. The AMD product appears to work just fine, and at $99 the cost is well within most budgets.

Extended Memory

In the PC world of IBM based machines, the 640K memory space has always been a problem. Two ways around this limit have been use of *expanded* memory and now *extended* memory. Expanded memory is the extra memory beyond the 640K but not beyond the 1Meg. This is also called shadow memory by some, as it resides behind the ROM space. Accessing expanded memory or boards that use expanded memory addressing is rather easy and free of problems.

Extended memory is the memory space beyond the 1 Megabyte limit. Using this memory unfortunately is not as easy as expanded memory. The only way to get to it in a normal PC is to use a 286/386 in protected mode. The DOS system interrupt $15 and it's function $87 provide a means of getting to it. The documents have a big warning however that *no* interrupts are allowed during data moves.

At my work we have two board designs; one uses an dual ported memory device located in the PC option ROM address. Writing to that board is easy, fast, and can interleave with interrupts without problems. The second design however uses memory mapping in extended memory. Problems abound with that design and we have only been able to barely make it work. Extended memory and interrupts currently do not work together. OS2 gets by the problem by putting the machine in protected mode all the time. What we needed is a protected mode DOS operating system.

I have checked the actual IBM ROM code and found that the INT $15, function $87 puts the machine into protected mode, and then when finished, does the same as an CLT-ALT-DEL, or master reset of the entire CPU. The WARM reset checks a CMOS memory location (that is where all the configuration information is kept) and based on what is stored there, what type of reset action is needed. If you were to write your own program, as a FORTH friend of mine did, the only way out of protected mode he could find was a power–on reset.

In protected mode, the 286/386 uses different tables that contain the address for interrupts, program pages, and everything. The interrupt $15 has some ROM interrupt tables to allow it to get through the block move you want to do. For other interrupts however, these special tables don't exist. What would happen with interrupts without these tables is most likely disastrous. Add to that the fact that the actual address lines are turned off (A20 to A24) except in the protected mode, and one asks the question, what good is extended memory if you can only use it in protected mode.

Our company has decided that our only option is to redesign the board to not use extended memory. We think we can do this by using a PAL in place of an address buffer. The main problem is changing all past boards and software. This is an good example of where going to the latest and greatest

Continued Page 37

SAGE MICROSYSTEMS EAST

Selling & Supporting the Best in 8-Bit Software

- Automatic, Dynamic, Universal Z-Systems
  - Z3PLUS: Z-System for CP/M-Plus computers ($70)
  - NZCOM: Z-System for CP/M-2.2 computers ($70)
  - ZCPR34 Source Code: if you need to customize ($50)
- ZSUS: Z-System Software Update Service, public-domain software distribution service (write for a flyer with full information)
- Plu*Perfect Systems
  - Backgrounder ii: CP/M-2.2 multitasker ($75)
  - ZSDOS/ZDDOS: date-stamping DOS ($75, $60 for ZRDOS owners)
  - ZSDOS Programmer's Manual ($10)
  - DosDisk: MS-DOS disk-format emulator, supports subdirectories and date stamps ($30 standard, $35 XBIOS BSX, $45 kit)
  - JetFind: super fast, extemely flexible text file scanner ($50)
- ZMATE: macro text editor / customizable wordprocessor ($50)
- PCED — the closest thing to ARUNZ and LSH (and more) for MS-DOS ($50)
- BDS C — including special Z-System version ($90)
- Turbo Pascal — with new loose-leaf manual ($60)
- SLR Systems (The Ultimate Assembly Language Tools)
  - Z80 assemblers using Zilog (Z80ASM), Hitachi (SLR180), or Intel (SLRMAC) mnemonics
  - linker: SLRNK
  - TPA-based ($50 each) or virtual-memory (special: $160 each)
- ZMAC — Al Hawley's Z-System macro assembler with linker and librarian ($50 disk, $70 with printed manual)
- NightOwl (advanced telecommunications, CP/M and MS-DOS versions)
  - MEX-Plus: automated modem operation with scripts ($60)
  - MEX-Pack: remote operation, terminal emulation ($100)

Next-day shipping of most products with modem download and support available. Order by phone, mail, or modem. Shipping and handling $3 per order (USA). Check, VISA, or MasterCard. Specify exact disk format.

Sage Microsystems East
1435 Centre St., Newton Centre, MA 02159-2469
Voice: 617-965-3552 (9:00am - 11:30pm)
Modem: 617-965-7259 (pw=DDT) (MABOS on PC-Pursuit)